Real-time Human Interaction with Supervised Learning Algorithms for Music Composition and Performance

Rebecca Anne Fiebrink

A Dissertation
Presented to the Faculty
of Princeton University
in Candidacy for the Degree
of Doctor of Philosophy

Recommended for Acceptance
by the Department of
Computer Science
Adviser: Perry R. Cook

January 2011

Abstract

This thesis examines machine learning through the lens of human-computer interaction in order to address fundamental questions surrounding the application of machine learning to real-life problems, including: Can we make machine learning algorithms more usable and useful? Can we better understand the real-world consequences of algorithm choices and user interface designs for end-user machine learning? How can human interaction play a role in enabling users to efficiently create useful machine learning systems, in enabling successful application of algorithms by machine learning novices, and in ultimately making it possible in practice to apply machine learning to new problems?

The scope of the research presented here is the application of supervised learning algorithms to contemporary computer music composition and performance. Computer music is a domain rich with computational problems requiring the modeling of complex phenomena, the construction of real-time interactive systems, and the support of human creativity. Though varied, many of these problems may be addressed using machine learning techniques, including supervised learning in particular. This work endeavors to gain a deeper knowledge of the human factors surrounding the application of supervised learning to these types of problems, to make supervised learning algorithms more usable by musicians, and to study how supervised learning can function as a creative tool.

This thesis presents a general-purpose software system for applying standard supervised learning algorithms in music and other real-time problem domains. This system, called the Wekinator, supports human interaction throughout the entire supervised learning process, including the generation of training examples and the application of trained models to real-time inputs. The Wekinator is published as a freely-available, open source software project, and several composers have already employed it in the creation of new musical instruments and compositions.

This thesis also presents work utilizing the Wekinator to study human-computer interaction with supervised learning in computer music. Research is presented which includes a participatory design process with practicing composers, pedagogical use with non-expert users in an undergraduate classroom, a study of the design of a gesture recognition system for a sensor-augmented cello bow, and case studies with three composers who have used the system in completed artistic works.

The primary contributions of this work include (1) a new software tool allowing real-time human interaction with supervised learning algorithms, which includes a novel "playalong" interaction for generating training data in real-time; (2) a demonstration of the important roles that interaction—encompassing both human-computer control and computer-human feedback—can play in the development of supervised learning systems, and a greater understanding of the differences between interactive and conventional machine learning contexts; (3) a better understanding of the requirements and challenges in the analysis and design of algorithms and interfaces for interactive supervised learning in real-time and creative problem domains; (4) a clearer characterization of composers' goals and priorities for interacting with computers in music composition and instrument design; and (5) a demonstration of the

usefulness of interactive supervised learning as a creativity support tool. This work both empowers musicians to create new forms of art and contributes to a broader HCI perspective on machine learning practice.

Acknowledgements

I am indebted to the support, guidance, and inspiration of many people, without whom my research and this thesis would not be possible. I am tremendously grateful to have had the opportunity to work with my advisor, Perry Cook, who has tirelessly provided insight, encouragement, and inspiration throughout my graduate study at Princeton. Perry's guidance and the creative incubation of the Sound Lab have been critical in shaping the way I think about research, and I've had an immense amount of fun as one of his students.

I also owe many thanks to Dan Trueman, whose own work with the Wekinator has been instrumental in turning it into a useful tool, and whose vision and philosophy have informed much of the way I think about music and technology's role in its creation. Dan has happily participated in just about every role possible in the creation of a PhD thesis, including mentor, reader, critic, collaborator, and test subject.

I'd like to express my enthusiastic gratitude for my other reader, Dan Morris, for his hard work providing many invaluable suggestions and insights throughout the writing of this dissertation. As my reader, and previously as my mentor at Microsoft Research, Dan's input has stretched my imagination and challenged my own perspectives on HCI and music in many helpful ways. His input on this thesis, along with that from Perry and the first Dan, has led to a much stronger document than I could have created on my own.

I am very appreciative of the work of my other committee members, Adam Finkelstein, Ken Steiglitz, and "alternate" Szymon Rusinkiewicz. I have greatly benefited from their enthusiasm for my research, their thought-provoking questions, and their decision not to make me do any more work.

The work in this thesis has been graciously supported by a number of sources. This material is based upon work supported under a National Science Foundation Graduate Research Fellowship.[1] I am also grateful for the support of the Francis Lothrop Upton Fellowship, National Science Foundation grants 0101247 and 0509447, and grants from the Kimberly and Frank H. Moss '71 Research Innovation Fund (Princeton School of Engineering and Applied Sciences), the David A. Gardner '69 Magic Project (Princeton Council of the Humanities), and the John D. and Catherine T. MacArthur Foundation.

I am incredibly appreciative of everyone else who has played a part in making the Wekinator software what it is and in making this thesis possible. To all the composers who participated in the "Music and Machine Learning" seminar in 2009—Anne Hege, Cameron Britt, Dan Trueman, Konrad Kaczmarek, Michael Early, Michelle Nagai, and MR Daniel—thank you for partaking in the adventure; for sharing your many ideas, questions, and complaints; and for the chance to create music together. To all the COS/MUS 314 (PLOrk) 2010 students: thank you for bringing your enthusiasm and creativity to your work with the Wekinator and throughout the course; it was a pleasure to teach you, and I learned a lot from working with you. To Meg Schedel: thanks for your vision, your Max/MSP expertise, your many ideas for improving the

[1] Any opinions, findings, conclusions or recommendations expressed in this publication are those of the author and do not necessarily reflect the views of the National Science Foundation.

Wekinator, your patience when things crashed, and your hospitality. To Raymond Weitekamp, as well as Michelle Nagai and Dan Trueman (who deserve to be thanked multiple times): thank you for the leap of faith you took in using the Wekinator in live performance, for your thorough and thoughtful feedback on the software, and for your music. I'm honored to have played a part in your work.

A tremendous thanks goes to Ge Wang for his work as the creator of the most-awesome ChucK language, as well as for being my collaborator and friend. ChucK has made my software development and teaching a joy, and the Wekinator has greatly benefited from its inclusion. Ge's collaboration on creating UAnae was integral to allowing the Wekinator to do live audio analysis, and our work together on SmirK provided the initial inspiration for this thesis.

The Wekinator would not be possible without the work of Adrian Freed and Matt Wright, whose invention of Open Sound Control has made development across different languages and hardware devices a cinch (most of the time). I must also thank Tom Lieber, who in five minutes hacked together the webcam edge detection module that has persisted as one of the most fun and demo-able feature extractors in the Wekinator, and Nikolaus Gradwohl, whose color-tracker I adapted for use in the Wekinator, leading to the endless amusement of many of my students and friends.

I'd like to thank Barry Threw and Keith McMillen Instruments for helping to make my work on bow gesture recognition possible. Barry has provided invaluable support with the K-Apps software. Keith McMillen and his company have provided a very fun and useful tool in the from of the K-Bow, and they have graciously given me permission to reproduce some of their photos and diagrams.

I am grateful to Jeff Snyder for some great brainstorming conversations and for taking several of the photos that appear in this thesis; I am also thankful to Jeff and to Cameron Britt for helping to lead an army of PLOrk students through their first steps with the Wekinator. I am indebted to the efforts of Andrew McPherson: his extensive feedback on the Wekinator software was not explicitly discussed in this thesis, but it has helped to shape the current and future form of the Wekinator in many valuable ways.

The work presented in this thesis has also been significantly shaped by inspiration and knowledge I've received through discussions with many other researchers and musicians, including Ashley Burgoyne, Ben Knapp, Betsey Biggs, Cory MacKay, Dave Benson, David Blei, Doug Eck, Eric Nichols, Georg Essl, Jason Hockman, Joel Chadabe, Laetitia Sonami, Phoenix Perry, Rob Schapire, Roger Dannenberg, Samson Young, Scott Smallwood, Seth Cluett, and Sumit Basu. I am also very grateful for the feedback I received at the CHI 2010 Doctoral Consortium, and for my discussions with Kayur Patel, Per Ola Kristensson, Wendy Kellogg, and many others.

I am grateful to have had the opportunity to work with many people outside academia who have taught me many valuable lessons about research, software engineering, innovation, and the "real world." I have benefited immensely from my work and discussions with Dan Morris, Merrie Morris, Sumit Basu, and countless others at Microsoft Research; from the mentorship and collaboration of Paul Lamere and many others during my internships at Sun Labs; from the chance to work with a fantastic

team of innovators at Smule; and from the vision and knowledge of Jay LeBoeuf, Stephen Travis Pope, and the rest of the team at Imagine Research.

I owe much credit for my success in graduate school to my Master's advisor, Ichiro Fujinaga. He taught me an enormous amount about research, and I am grateful to have had the opportunity to be introduced to the computer music research field by him and by my other instructors at McGill, including Gary Scavone, Marcelo Wanderley, Philippe Depalle, and Stephen McAdams.

My time at Princeton has been made immeasurably more productive and enjoyable by many members of the university community. In addition to everyone mentioned previously, huge thanks go to Alicia Juskewycz, Ananya Misra, Matt Hoffman, Sonya Nikolova, and Xiaojuan Ma for their friendly insight and support; to Jeff Dwoskin for the use of his LaTeXtemplate; to Melissa Lawson for taking care of all us CS graduate students; and to the CS department staff for making everything work. My time at Princeton has been enriched by the members of the machine learning reading group, the Princeton Laptop Orchestra, SideBand, the LGBT Center, the Queer Graduate Caucus, and the UU Campus Community, as well as the tireless staff of Hoagie Haven.

I would also like to express my sincere appreciation for the Computer Science Department faculty; as a student, I have benefited from their guidance and from their efforts to create a challenging and supportive environment for their students; as a near-graduate, I am honored by their confidence in my abilities and in the value of my work.

A heartfelt thanks goes to my past teachers and mentors—Mona Grote, Jane Kutcher, Beth Owen, Greg Taylor, Katherine Borst Jones, Nicole Molumby, Tim Long, Bettina Bair, David Huron, and all the others—for the lessons they have taught me about music, about computer science, and about research, and for the inspiration and support they have offered as I've strived to make all of these a part of my life.

I must express my deep appreciation for all my friends who have helped me to become who I am. I am lucky to have too many of you to name.

Finally, thank you to my family—to Mom, Dad, Stephanie, my grandparents, my parents-in-law, and everyone else. I am incredibly fortunate to have your constant love and support. And, of course, thank you to Ariane, for your incredible patience, brainstorming, encouragement, and love; you are simply the best.

To my parents, Mark and Dianne Fiebrink

Contents

List of Tables

List of Figures

Chapter 1

Introduction

> "The old computing was about what computers could do; the new computing is about what users can do."
>
> —Ben Shneiderman (2002, 2)

1.1 An HCI Perspective on Machine Learning

Machine learning offers a set of powerful algorithmic tools for understanding, modeling, and making decisions from data. Application of these tools through recent decades has led to advances, if not revolutions, in domains as diverse as bioinformatics, information retrieval, gaming, robotics, and beyond. It is the charge of machine learning research to improve this palette of tools by developing algorithmic techniques that are increasingly faster, more accurate, and applicable under a broader array of problem structures and constraints.

Research in the field of human-computer interaction (HCI), in contrast, is concerned with the broader human context of computing systems. A significant aim of HCI research is to develop technology and practices that improve the *usability* of computing systems, where usability encompasses the effectiveness, efficiency, and satisfaction with which the user interacts with a system (ISO 9241-11:1998 1998). Even though human application of machine learning algorithms to real-world problems requires embedding the algorithms in software or hardware tools of some sort, and the form and usability of these tools impact the feasibility and efficacy of applied machine learning work, research at the intersection of HCI and machine learning is still a relatively young area.

What might be the consequences of examining machine learning through the lens of human-computer interaction, and in particular, of enabling machine learning users to exercise a richer array of interactions with machine learning algorithms? Can we make machine learning more usable, thereby enabling human users' work with these algorithms to be more efficient, more effective, and more satisfying? Can we come to a better understanding of how machine learning algorithms and interfaces impact not just the speed and accuracy of users' work, but also how users think about their work and formulate their goals? Can we find a use for machine learning algorithms in

unconventional contexts, such as the support of human creativity and discovery? Can we design more effective human workflows for building machine learning systems, enable successful application of algorithms by machine learning novices, and ultimately make it possible in practice to apply machine learning to new problems? These are the fundamental questions motivating this thesis.

1.2 Supervised Learning and Computer Music

The work in this thesis concerns a family of machine learning and pattern recognition algorithms known as supervised learning. A supervised learning algorithm is essentially a tool for producing a mathematical model or function that, given some input, produces some output. The algorithm infers, or *learns*, this function from a set of training data, which consists of a collection of example inputs paired with their corresponding outputs. Supervised learning algorithms are designed to be capable of modeling complex relationships between inputs and outputs while retaining the ability to generalize, producing reasonable outputs for new inputs not present in the training set.

Computer music is a domain rich with computational problems requiring the modeling of complex phenomena, including the relationships between low-level audio and sensor signals and higher-level analyses of pitch, harmony, instrumentation, human gesture and intention, and even emotion. The ability to accurately model these complex relationships has far-reaching implications for transforming not only live performance, but also music composition, studio production, sound design, the design of new musical instruments, and content-based music search and recommendation. Supervised learning is an attractive tool for dealing with complex musical phenomena across all these problems and application domains, and it has been applied to many such problems in music. For example, to build a pitch classifier capable of transcribing the melody from an audio file, one might train a learning algorithm on a dataset of recorded songs that have each been explicitly annotated with their melodic pitch content over time, as in the work of Ellis and Poliner (2006). Or, to build a system in which a human performer wearing a sensor glove can "play" a computer synthesis algorithm in real-time, one could train an algorithm on the glove sensor outputs paired with corresponding synthesis parameter values for different hand gestures, as in work by Modler et al. (1998).

Many computer music applications possess additional characteristics, beyond the requirement for dealing with complexity, that make them suitable to the application of supervised learning algorithms. As in our gesture glove example, computer music performances often incorporate significant real-time interactivity, wherein a computer's actions are driven by real-time analysis of live, human-generated signals. Also, music is a creative domain, and the creativity of composers or performers can be enhanced or inhibited by the hardware and software tools they use to do their work. As we will show, supervised learning can be a particularly effective tool for building real-time interactive systems and for supporting creative work.

The scope of this dissertation is therefore the study of human-computer interaction with supervised learning algorithms in computer music application domains. While supervised learning has been applied to problems in computer music before, most prior research has focused on seeking new algorithmic approaches to better solve particular computational problems. We employ a new, complementary perspective in this research; by focusing on questions of improving usability, leveraging interaction more effectively, and better understanding the ways in which machine learning algorithms and interfaces impact users' work, this research opens important new avenues for intellectual discovery as well as opportunities for users to apply supervised learning more effectively and in new ways.

1.3 Goals and Contributions

The goals of this work have been twofold: first, we have endeavored to make supervised learning algorithms more usable by practicing musicians, enabling them to work more effectively and empowering them to create new forms of art; second, we have aimed to gain a deeper knowledge of the human factors surrounding the application of supervised learning to musical problems, and in the process contribute to a broader HCI perspective on machine learning practice.

Prior to this work, there existed no software tool which met the infrastructure, interaction, and algorithmic needs of a broad set of musicians and researchers desiring to apply supervised learning to musical and real-time problems. Therefore, we have built a new, general-purpose software system for applying supervised learning in music and other real-time and creative problem domains. This system, called the Wekinator, is unique in the breadth of applications it supports as well as the human-computer interactions it enables. The Wekinator provides user interfaces for the construction, evaluation, and real-time application of standard supervised learning algorithms. The system emphasizes the role of human interaction throughout the entire supervised learning process, including the creation and modification of training datasets and the evaluation of trained models. Among the interactions supported by the Wekinator is a novel "playalong" interaction that enables users to create training data by gesturing along with previously-annotated performance sequences. The Wekinator is compatible with a widely-used machine learning toolkit called Weka (Hall et al. 2009), and its architecture allows it to be easily incorporated into real-time systems implemented in nearly any programming language, in both musical and non-musical domains.

We have also used the Wekinator to study human-computer interaction with supervised learning in several musical applications with different types of users. These users have been composers writing new music and designing new interactive musical instruments, students studying interactive systems-building in an undergraduate computer music course, and a composer/cellist developing a gesture recognition system for a sensor-augmented cello bow. Through this work, we have demonstrated the breadth and importance of the roles that interaction—encompassing both human-computer control and computer-human feedback—can play in the development of

supervised learning systems, and we have achieved a greater understanding of the differences between interactive and conventional machine learning contexts. We have also gained a better understanding of the requirements and challenges in the analysis and design of algorithms and interfaces for interactive supervised learning in real-time and creative problem domains.

Additionally, practicing composers and instrument designers have rarely been studied in their workplace interactions with technology, and our work with these users has led to a clearer characterization of the human-computer interaction requirements relevant to composition and instrument design. Finally, our observations of musical users engaged with the Wekinator have increased our understanding of how interactive supervised learning can also function—outside its conventional usage—as a creativity support tool.

We have previously published some of the work presented in this thesis. Specifically, we have described an early version of the Wekinator software in Fiebrink, Trueman, and Cook (2009); described our playalong method of training data creation in Fiebrink, Cook, and Trueman (2009); discussed the participatory design process described in Chapter 4 and its outcomes with regard to improvements to the Wekinator and a deeper understanding of HCI in composition in Fiebrink, Trueman, et al. (2010); presented preliminary work on the bow gesture recognition project of Chapter 6 in Fiebrink, Schedel, and Threw (2010); and presented a summary of the thesis work and goals in Fiebrink (2010).

1.4 Outline

In the following chapter, we provide an overview of supervised learning and computer music to acquaint the reader with relevant background in these fields. We also highlight three problems within computer music where supervised learning is particularly useful: the "mapping problem," gesture recognition, and semantic audio analysis. Following an overview of existing tools for applying machine learning in music and in other domains, we motivate the need for a new software tool for interactively applying machine learning to real-time signals. We present an overview of prior HCI research that has investigated the questions of how to make machine learning more usable, and how to more effectively engage human interaction in machine learning practice. We also briefly introduce the thread of HCI research that studies the use of computers to support human creativity.

In Chapter 3, we present our new software system, the Wekinator, which we have created to better support the application of supervised learning to music and real-time domains, and which we have used throughout our studies of users in this thesis. We describe the interactive workflow it supports, walk through an example use case, and provide a detailed overview of the system and its user interfaces.

In Chapters 4 through 7, we present four studies in which we have applied the Wekinator to researching human-computer interaction with supervised learning in particular computer music composition and performance scenarios. These studies include a participatory design process with practicing composers, pedagogical use

with non-expert users in an undergraduate classroom, a study of the design of a gesture recognition system for a sensor-augmented cello bow, and case studies with three composers who have used the system in completed artistic works. In each chapter, we provide background on the application domain, describe the study goals and method, and present the study results. In each case, we discuss the results with particular attention to an analysis of users' goals in their work with the Wekinator and the actual outcomes of this work, observed patterns of user interaction with the system, an evaluation of system usability, and implications for future work.

In Chapters 8 and 9, we further examine the most significant findings across our four studies. Chapter 8 focuses on the roles that interaction played in different stages of users' work with supervised learning, from training set creation to evaluation. We discuss the implications of our observations regarding training set size and evaluation methods in interactive supervised learning. We also discuss their implications regarding the requirements and challenges in the analysis and design of algorithms and interfaces for interactive supervised learning, with attention to how the characteristics of the computer music problem domain shaped the types of interactions that were possible for the users we observed.

In Chapter 9, we argue that interactive supervised learning can function as an effective creativity support tool. We examine users' interactions with the Wekinator in the context of prior work on creativity support tools and embodied cognition, and we demonstrate how interactive supervised learning serves several important creative functions. We also review how our observations of and work with Wekinator users contribute to a deeper understanding of the human-computer interaction requirements of composers, and we emphasize the utility of broadening the scope of discussion about interaction in computer music to include the processes of composition and instrument design.

We conclude in Chapter 10 with a review of our most significant findings and contributions. We describe our next steps for improving the Wekinator software, and we outline new research directions in HCI, machine learning, and computer music that our work has suggested may be fruitful in further enabling users to effectively apply supervised learning algorithms to real-world problems.

Chapter 2

Background and Motivation

This thesis draws heavily on prior work and ideas in computer music, human-computer interaction (HCI, also sometimes called CHI), and machine learning. In this chapter, we provide broad background information to acquaint the reader with each of these fields, and we define key ideas and terminology that will be used throughout this thesis. We start by providing an overview of computer music, discussing the roles that humans and computers can play in interactive computer music performance systems, and highlighting important computational problems in the design of interactive computer music software. Next, we provide an overview of supervised learning, the family of machine learning algorithms that we will consider in this thesis. We define key vocabulary relevant to this thesis, and we discuss how supervised learning algorithms are conventionally applied and evaluated. We then describe three major areas of computer music in which supervised learning is a useful tool: the creation of new digital musical instruments, recognition of human gesture, and semantic audio analysis. We describe existing tools for applying machine learning in music and motivate the creation of a new tool that may better meet the needs of more users applying supervised learning to more problems. Subsequently, we discuss recent research in HCI that seeks to leverage human interaction in new ways in the creation of machine learning systems, as well as HCI research studying how technology can support people doing creative work, and we describe how our work fits into these contexts. We conclude the chapter with a review of the key findings of the prior work that provides the foundation for this thesis, followed by a statement of our research motivations in creating and studying our new tool for interactive supervised learning in music.

2.1 Computer Music

2.1.1 A Brief History

The use of computers to compose and play music dates back approximately sixty years. According to Doornbusch (2004), it is possible that the first use of the computer to make music was in 1950 or 1951, when the Australian CSIR Mk1 computer was programmed to play simple melodies by a musically-inclined mathematician named Geoff Hill. Around the same time, the British computer scientist Christopher Strachey

programmed the Ferranti Mark 1 machine at the University of Manchester to play music. That machine's renditions of classic melodies were recorded by the BBC in the autumn of 1951 (Fildes 2008). A few years later, a researcher at Bell Labs named Max Mathews developed MUSIC, the first programming language for creating music and audio (Chadabe 1997, 109).

Even given the limited processing power, expense of operation, and other inconveniences of working with mainframe computers, pioneers such as Mathews and his collaborators at Bell Labs recognized the potential of the computer to make wholly new types of music and sound. By the 1950s, contemporary composers such as Pierre Schaeffer and Karlheinz Stockhausen were already using recording technologies and analog electronics to create music that was unconstrained by the sounds, performance practices, and conventions of acoustic music (Doornbusch 2004). Composers (albeit only those with the means and access to program the computers of the time) began to employ digital technology to similarly push the boundaries of musical practice, following Mathews' creation of MUSIC in 1957. For example, James Tenney, Jean-Claude Risset, and Charles Dodge were among the composers at Bell Labs and Princeton University who used MUSIC and its descendants (the MUSIC-N family of programs) throughout the 1960s to create new compositions that explored the computer's musical possibilities (Chadabe 1997, 114).

Computer music has grown dramatically in the past sixty years as an area of academic inquiry and artistic pursuit. With the advent of minicomputers and then personal computers, many more musicians and researchers gained access to the means to develop their own musical software and computer music compositions. A host of programming languages and environments such as Max (Puckette 1991, now Max/MSP) and CSOUND (Boulanger 2000) evolved to meet the needs of these users. Real-time control over digital sound became possible through the use of specialized DSP workstations such as the IRCAM Signal Processing Workstation (Lindemann et al. 1990), as well as digital synthesizers such as the Yamaha DX-7 (Reid 2001); digital music thus became something that one could *perform* live. Composers and technologists began to envision, build, and use new mechanisms for controlling the computer in real-time, ranging from manipulating GUIs (such as those in Max), to interfaces like the MIDI keyboard which closely resembled conventional instruments, to acoustic instruments augmented with sensors (such as the hypercello of Paradiso and Gershenfeld 1997), to entirely new sensor-based gestural interfaces (such as the *The Hands* by Waisvisz 1985). Increasing processing power has enabled the real-time execution of complex software programs for accompanying and improvising alongside human musicians (e.g., see work by Raphael 2010 and Kapur 2007), while the shrinking size of computing systems has enabled increasingly portable and populous computer music performance ensembles like the Princeton Laptop Orchestra (Trueman et al. 2006) and the Stanford Mobile Phone Orchestra (Wang et al. 2008).

The work in this thesis focuses primarily on the use of computers in academic and art music, a field that traces its roots to the early work of Mathews, Risset, Tenney, Dodge, and others. The question of how technology can enable new means of musical expression remains central to the computer music field today, and it is a fundamental question explored in this thesis. The relevance of this question is not at all limited

to academic or art music, however, as the use of technology has also become central to mainstream and popular music of many genres. For example, numerous popular music artists have employed digital technology in their performances for many of the same reasons as computer and electronic art composers and performers, including to broaden their palette of sounds and to expressively control digitally-produced sound in new ways. Hardware digital synthesizers were adopted by popular musicians in the 1980s and onwards, and since the 1990s, software synthesis and and audio effects modules have become widely used in both studio production and live performance. The introduction of the MIDI communication protocol in the early 1980s (Chadabe 1997, 195–196) enabled performers to play a variety of synthesizers using standard MIDI keyboards, foot controllers, and drum triggers. These controllers, as well as alternative hardware interfaces such as the Monome (Dunne 2007) and Lemur (JazzMutant 2009), are used not just as instruments, but also to control high-level musical processes (e.g., to trigger sequences of rhythms in a drum machine), and to expressively control parameters of audio effects applied to the musicians' live sound.

Some computer music performance technologies developed in academic contexts also end up being adopted by mainstream artists. For example, John Chowning discovered the FM synthesis algorithm while working at Stanford University in 1967 (Chowning 1973). Packaged into the Yamaha DX7 synthesizer in 1983, FM synthesis enabled countless artists of the 1980's to efficiently create a wide range of artificially synthesized sounds (Reid 2001). More recently, Max/MSP has been used by the English alternative rock band Radiohead (Pareles 2007), and the Reactable tabletop interface (developed at the Universitat Pompeu Fabra by Jordà et al. 2007) was used by the Icelandic singer-songwriter Björk on her 2007 world tour (Andrews 2007).

2.1.2 Interactive Computer Music

Definition and Scope

Computer music involves a wide range of live performance practices. One significant dimension of variability among these practices is the extent to which human performers exercise control or influence over the live actions of the computer. At one extreme of this spectrum of human involvement lie practices such as tape music and algorithmic computer music, in which the sounds produced by the computer are uninfluenced by human performers, or in which there are no human performers at all. At the other extreme lie practices in which the computer is highly responsive to human actions. For example, a human performer might control the pitch, articulation, volume, and timbre of a computer synthesis algorithm through her gestures using a hardware controller. In this case, the controller and computer software together function as an expressive musical instrument. Work mentioned above by Waisvisz (1985) falls into this category, as do notable works by Laetitia Sonami (Bongers 2000), Trueman and Cook (2000), and many others. Alternatively, the computer might listen to the sound of a human musician performing on an acoustic instrument and respond by producing its own musically appropriate output. In this case, the computer may play a role more akin to a human accompanist or collaborator. Notable work designing collaborative

and accompanying computer music systems has been done by Dannenberg (1989), Lewis (2000), Rowe (1993), Raphael (2010), and others.

In this thesis, we focus primarily on computer music in which human performer(s) exercise a high degree of performance-time control and influence over the computer(s). Within computer music performances of this nature, there exists a great diversity in the types of roles played by the computer, and in the ways that humans exercise (and experience) control and influence over the computer's behavior. For example, Lippe (2002) writes about the role of the computer: "The computer can be given the role of instrument, performer, conductor, and/or composer. These roles can exist simultaneously and/or change continually..." The computer may produce sound by digital synthesis, by applying live processing to the sound of the performer, or by a mixture of these and other means. Simultaneously, a performer might control low-level properties of the computer process at a fine granularity, or he might control higher-level structural properties or influence the outcome of algorithmic processes. The performer's intentions may be explicitly concentrated on manipulating the computer, or he may be focused on affecting other aspects of the performance, without attention to how his actions are interpreted or acted upon by the computer. The computer might sense the actions of the human through audio, sensors, or other means; and the method by which the computer translates human actions into control parameters driving its own synthesis algorithms or other processes may remain fixed throughout a performance, or it may change over time.

Within this broad context, composers have offered different definitions of what constitutes *interactive computer music*, as Lippe (2002) discusses:

> Robert Rowe, in the seminal book *Interactive Music Systems* (Rowe 1993), states: "Interactive music systems are those whose behavior changes in response to musical input." A dictionary definition of the word interactive states: "capable of acting on or influencing each other." This would imply that a feedback loop of some sort exists between performer and machine. Indeed, the Australian composer Barry Moon suggests: "levels of interaction can be gauged by the potential for change in the behaviors of computer and performer in their response to each other" (Moon 1997). George Lewis, a pioneer in the field of interactive computer music, has stated that much of what passes for interactive music today is in reality just simple event "triggering," which does not involve interaction except on the most primitive level. He also states that since (Euro-centric) composers often strive for control over musical structure and sound, this leads many composers to confuse triggering with real interaction. He describes an interactive system as "one in which the structures present as inputs are processed in quite a complex, multi-directional fashion. Often the output behavior is not immediately traceable to any particular input event (Rowe et al. 1993). David Rokeby, the Toronto-based interactive artist, states that interaction transcends control, and in a successful interactive environment, direct correspondences between actions and results are not perceivable. In other words, if performers feel they are in control of (or

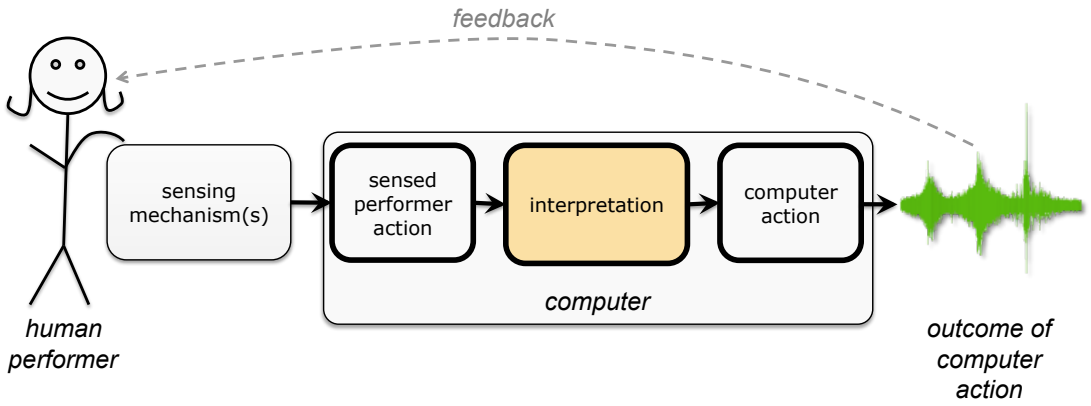


Figure 2.1: In interactive computer music, actions of a human performer are sensed in real-time by a microphone, controller, and/or other sensing mechanism, and communicated to the computer. The computer interprets these actions, and this interpretation is used to control or influence its future actions. The outcome of the computer action—for example, changes in the timbre of its sound or in the type melodic material it generates, provides real-time feedback to the human performer.

> are capable of controlling) an environment, then they cannot be truly interacting, since control is not interaction.

In this thesis, we will consider a broad definition of interactive computer music systems, which minimally meets Rowe's definition of changing their behavior in response to musical inputs by a performer, but which also incorporates the possibility for Lewis's notions of complex and multi-directional systems of influence, as well as the potential for transcending ideas of "control" according to Rokeby. Each of these definitions, as well as the diversity of performance scenarios described earlier, can be discussed in terms of the system components pictured in Figure 2.1. Fundamentally, an interactive computer music system incorporates one or more mechanisms by which the computer senses information about the actions of a performer, interprets this information, and takes some action based on this interpretation (for example, the setting of a synthesis parameter, or a change in tempo, or the creation of new motivic material). The actions of the computer are conveyed to the performer (in the simplest case, by the performer listening to the computer's sound output), and depending on the compositional constraints of the piece, the performer may subsequently interpret and respond to the computer's actions.

Composition in Interactive Computer Music

The role of the composer in interactive computer music therefore involves a much larger scope than in traditional contexts where composition involves the specification of how sounds will be played over time. The creation of an interactive composition involves choosing among the many possibilities discussed above regarding the roles of

the computers and humans and the computer mechanisms for sensing and responding to human actions, as well as designing the computer's processes for generating sound, and potentially specifying the way that all of these variables will change over time.

Composer's writings provide insight into how they have navigated through this large set of compositional possibilities in the process of composing interactive computer music. Work by Hahn and Bahn (2003), Lewis (2000), Lippe (1997), Trueman and Cook (2000), and many others underscores the diversity of approaches taken to computer music composition practice. Composers' reflections on their practice also highlight the importance that they place (and the effort they expend) on crafting the interaction between humans and computers, as the nature of this interaction can be integral to the identity of the piece itself.

We note that, when the computer takes the role of an instrument within an interactive computer music context, the process of designing how a performer will use the computer to play sound can be understood as both composition and instrument building, or—to borrow a term from Schnell and Battier (2002)—*composing* the instrument. This process entails the design of how the computer will sense the performer's actions, the design of how the computer will produce sound (e.g., using a synthesis algorithm), and how the computer will *map* from the actions of the performer into influence over the produced sound. While some interactive computer systems are designed to be played across a variety of compositions written by different composers (i.e., to function as conventional instruments do), it is also common for composers to design new instruments intended only for use in a particular composition, or to compose instruments in which there is no clear dichotomy between instrument and composition (see, e.g., Cook 2001). In this thesis, we will not strongly distinguish between creating compositions, interactive systems, and instruments: to the extent to which these can even be considered to be separate tasks, we will be focusing on computational and interactive challenges that are common across them all.

Computational Problems in Interactive Computer Music

While the many design decisions involved in the composition of an interactive computer music system may, of course, be strongly influenced by composers' aesthetic concerns and compositional goals, they are also influenced by computational and technical concerns. In particular, creating computer systems with the ability to accurately sense, interpret, and act on human actions touches on some difficult computational problems. First of all, computer analysis of human gesture musical audio can be computationally challenging, as there may exist a large "semantic gap" (Lew et al. 2006) between the low-level gestural or audio signals and the semantically- or musically-relevant interpretations of those signals (such as a performer's cues, harmony, or intentions) upon which the composer would like the computer to act. Such analysis problems overlap with significant ongoing research threads in human-computer interaction, computer vision, signal processing, and music information retrieval, among other domains.

Secondly, the creation of a musically satisfying and engaging interactive system may necessitate complex relationships between performer actions and computer re-

sponses. Such complexity can be motivated both by composers' compositional and aesthetic values (see the above discussion on Lewis and Rokeby, for example), and by empirical results. For example, Hunt and Kirk (2000) created a complex musical control interface in which changes in each dimension of performer gesture affected multiple sound synthesis parameters and each sound synthesis parameter was affected by multiple gestural dimensions. Comparing this interface to simpler ones, they observed that the more complex interface allowed participants to perform musical control tasks more accurately, that it enabled participants' accuracy to improve more with practice, and that it was also more fun to use. The creation of systems incorporating appropriately complex relationships between performer actions and computer responses therefore necessitates that composers and instrument designers have access to algorithmic tools for designing complex functions, and to user interfaces that enable them to apply these tools effectively and efficiently.

Supervised learning has been employed by composers and researchers as a tool to address these computational problems of interpreting complex signals and producing complex behaviors in interactive computer music. We will further discuss in Section 2.2.3 why supervised learning is useful for addressing these problems, after the following general introduction to supervised learning.

2.2 Supervised Learning

2.2.1 Overview and Terminology

This work concerns a family of machine learning and pattern recognition algorithms known as supervised learning. Here we will give only a very basic overview of supervised learning; a more thorough treatment can be found in Bishop (2007), which is the reference for our overview unless otherwise indicated. In the following discussion, key terms used throughout this thesis are printed in **boldface**.

A supervised learning **algorithm** is essentially a tool for producing a mathematical **model** or function that, given some input, produces some output. The algorithm infers, or *learns*, this model from a **training dataset**, which is a collection of **data points** (also called **"instances"** or **"examples"**) consisting of example inputs paired with their corresponding outputs, or **"labels"** (also sometimes called "targets"). The process of producing the model from the training set is called **training**. After this model is built, it can compute new output values for new inputs, even for inputs not present in the training set. Figure 2.2 illustrates the relationship between the data, algorithm, training, and model.

The inputs to a model are vectors of numbers, or **features** (also sometimes called **attributes**). For example, a model of human gesture might take in a feature vector extracted from sensors worn on the body, where the features themselves are raw sensor values, statistics computed on those sensor values, or both. In a **classification** problem, the output of the model is one of a finite set of discrete labels, also referred to as the set of **classes**. For example, a **classifier** algorithm could be used to build an American Sign Language fingerspelling classifier that outputs the letter a human

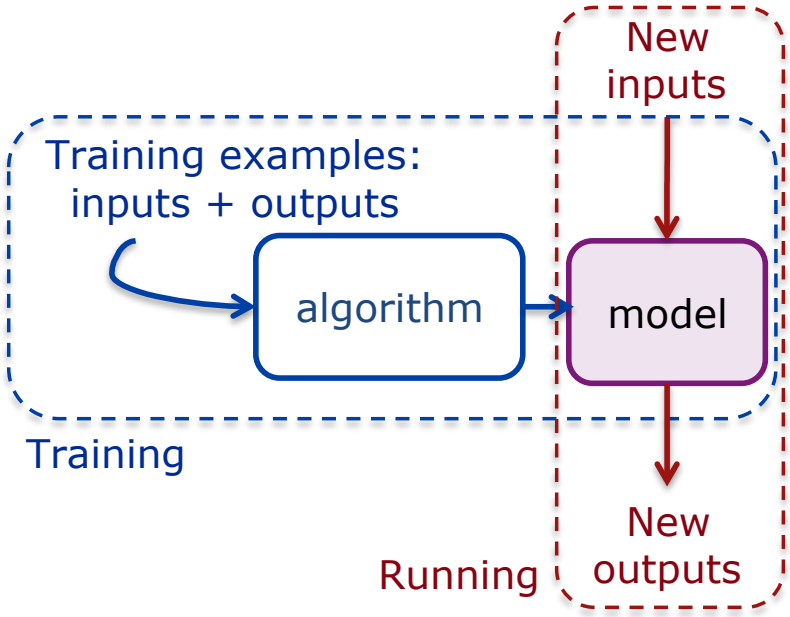


Figure 2.2: A supervised learning algorithm creates a model from training data, which consists of example inputs paired with corresponding outputs. The trained model can then compute new output values from new inputs.

is signing, as in work by Allen et al. (2003). In that case, the label set of the classifier would include the 26 letters of the alphabet[1].

On the other hand, in a **regression** problem, the model's output value is not constrained to be a member of a finite class set; it may take any real value. Regression could be used to build a continuous gesture-to-speech controller, such as one component of Fels and Hinton's Glove-TalkII system, in which continuous hand motions result in gradually changing articulatory control over a speech synthesis algorithm (1995). In this example, a regression model would compute the numeric value of a speech synthesis parameter from the glove sensor features. (It is also possible to build multidimensional regression models that output a vector of real numbers, but in this work we have used collections of unidimensional regression models instead of multidimensional models, for reasons we explain in the next chapter.)

Many different supervised learning algorithms have been designed to produce a models from a training set. In this work, we have primarily focused on the use of the AdaBoost, support vector machine, decision tree, and k-nearest neighbor algorithms for classification, and the multilayer perceptron neural network algorithm for regression. These algorithms are discussed in the next chapter, in Section 3.3.3.

Typically, the goal of any supervised learning algorithm is to model the relationship between inputs and outputs in the training set in a way that allows generalization, or producing reasonable outputs for new inputs not present in the training data (also

[1]In reality, Allen et al. (2003) used only 24 classes; they ignored the letters 'J' and 'Z,' whose fingerspellings require hand motion.

called "predictive performance"). Concern for generalization is at the core of the theoretical analysis and design of machine learning algorithms. In the PAC-learning framework, for example, a learning algorithm is by definition capable of classifying instances not (necessarily) in the training set with a high accuracy rate, with a high probability (Valiant 1984). Widely-used algorithms such as AdaBoost and support vector machines are designed to explicitly to maximize **generalization accuracy** (that is, the proportion of future inputs that are assigned the correct label) (Bishop 2007).

This primary emphasis on generalization accuracy underlies standard metrics used to evaluate the suitability of an algorithm (or parameterization thereof) for modeling a given dataset. Unless one knows the identity of all future inputs to a model and their proper labels (in which case building a perfect model is trivial), generalization accuracy must be estimated from the available data. Using accuracy on the model's training dataset is a poor estimate, as this can assign too favorable a rating to a model that has "overfit" to the training data and is poor at generalizing. So, the available data can be partitioned into a training set and a mutually exclusive (or "held-out") **test set** for evaluating performance. Generalization accuracy for a model trained only on the training set may then be estimated by computing accuracy on the test set.

Cross-validation is a commonly-used technique that repeats this procedure several times in order to provide a less noisy estimator of predictive performance. As the process is repeated, each available instance is present in the test set of a single iteration. For example, the available data may be partitioned into two equally sized sets (called **"folds"**), which we will refer to as A and B. First, the learning algorithm to be evaluated is trained on A only, and the trained model's accuracy on B is computed. Second, the learning algorithm is retrained on B only, and the trained model's accuracy on A is computed. The **cross-validation accuracy** is computed as the average accuracy of these two tests. Cross-validation accuracy can be computed with larger numbers of folds (10-fold cross-validation accuracy, in which the dataset is partitioned into ten sets, is particularly common in practice). In the general case of n-fold cross-validation, each of the n iterations uses a different single fold as the test set, and in each iteration, the training set consists of the $n-1$ folds not currently used as the test set.

Alternative evaluation measures such as F-measure, cost, precision, recall, and area under ROC may be used under certain circumstances, but in each case, the goal remains to estimate the model's relevant future behavior from the available, finite data. In this work, we focus primarily on the use of cross-validation as a estimator of generalization accuracy. In Section 8.5.2, we present a discussion interrogating the meaning and appropriateness of generalization accuracy measures in interactive machine learning systems used in practice.

2.2.2 Supervised Learning in Practice

Why Supervised Learning?

Supervised learning has been applied in myriad domains, from natural language processing to medical diagnosis to computer vision and beyond. It offers a powerful computational tool for building models in domains where it may be easier to access or create a training dataset than to explicitly design the model function. For example, the construction of a software program that identifies human faces in images can present a daunting problem for a programmer, who must discover the mathematical relationship between individual pixel values in an image and the identity of a human face, then encode this relationship in software. It may be much simpler and faster to present a supervised learning algorithm with a set of images known to contain certain faces, along with their labels, and have the algorithm learn the model from the data. Furthermore, a programmer designing the model by hand would have to to take into account how aspects of the classification problem are likely to vary, so that the model's output will be robust to changes in lighting conditions, face angle, or haircuts. By using a training dataset in which these characteristics are varied, a supervised learning algorithm may learn a model that is robust to these effects.

Note that, in practice, someone must still design **feature extractors** to extract features from the image pixels. (Feature extraction is also typically performed to compute features from the gestural or audio signals we will consider later in this chapter.) In order for supervised learning to work, at least some of the extracted features must provide relevant information to the classification problem of face identification. But it is not necessary that these features be noise-free, that they all be relevant, or that they be informative for all faces or all images. The problem of feature design is much simpler, therefore, than designing the modeling function by hand. Additionally, in application domains such as computer vision or audio analysis, prior research has identified many features that are likely to be useful across many analysis problems, and new machine learning systems can leverage this domain-specific knowledge and even existing software tools for extracting standard features.

Modes of Application

Across different application domains, supervised learning is commonly used as a tool in both data analysis and software engineering. In data analysis, people—often domain experts—use models trained from the data to provide insight into patterns in the data, or to help them make decisions. For example, Ross et al. (2003) are hematology researchers who trained classifiers to a classify a pediatric leukemia patient's risk group from a set of genetic features. In this work, they were able to demonstrate that the genetic features under consideration were accurate predictors of risk group (and therefore might be studied further to "provide new insights into the altered biology underlying these leukemias.") Furthermore, they demonstrated that classifiers trained on these features could be useful as a new diagnostic tool. In other work in industry, supervised learning algorithms are often applied to gain economic advantages from a company's data. For example, Witten and Frank (2005) write,

"Patterns of behavior of former customers can be analyzed to identify distinguishing characteristics of those likely to switch products and those likely to remain loyal." Such information can be used to provide special incentives targeted to customers likely to leave.

Supervised learning can also be integrated into software to provide enhanced functionality to end users. For example, the developers of an e-mail spam filter program may train a classifier based on their own extensive dataset of emails labeled as "spam" and "not spam." By embedding the trained classifier into the software that they sell, people who buy the software can immediately apply the trained filter to their email. Alternatively, software applications may collect data from a user (or the user's system) to train a model customized to that user. For example, the spam software may instead construct its training dataset by providing the user with an interface to flag incoming emails as "spam" or "not spam." A classifier trained on this dataset will be customized to detecting spam in the types of email messages typically received by that user. It is also possible to combine these two approaches, in which data customized to the user is added to a pre-existing training dataset. In this case, the user is still able to use a pre-trained model without doing any labeling himself, but as he adds labels, the model can evolve a customized behavior.

Supervised learning components are often embedded in software systems in such a way that offers little or no transparency or control to the end user. In the spam filter software example, the user may understand that he is helping to improve the performance of a pattern recognition system by providing labeled email examples, but he may not know how this algorithm works, and he may not have the ability to modify it through any other means (e.g., changing the learning algorithm, its parameters, or the features it uses to classify emails). In other systems, the machine learning component may be entirely hidden from the user. For example, the Amazon.com recommendation engine (Linden et al. 2003) uses machine learning to suggest new products to users based on what others have bought (i.e., based on what its set of models have predicted a user might buy), but the workings of the models themselves are hidden from users, and users can only influence the models indirectly, through their purchases and browsing behaviors. This type of approach to the use of machine learning systems contrasts significantly with "end-user machine learning," in which users of the software have a significant degree of direct control and influence over the training and evaluation of supervised learning models. For example, an end-user machine learning system might engage the user in interactively providing training examples to teach the computer to classify images of people (e.g., Amershi et al. 2010). In our work supporting and studying human-computer interaction with machine learning systems, we focus on this latter type of context, in which users are closely involved with the process of training supervised learning models.

Tools for Applying Supervised Learning

While machine learning researchers often do collaborate with researchers and practitioners in application domains, much work applying supervised learning in research, data mining, and software development is performed by domain experts themselves.

The success with which these researchers and practitioners have been able to apply supervised learning to so many domains is due in part to the availability of general-purpose software tools for applying supervised learning to arbitrary problems. Environments such as Weka (Hall et al. 2009) and RapidMiner (Rapid-I 2010, formerly "YALE"), and libraries such as LIBSVM (Chang and Lin 2001) and the Neural Network Toolbox (Beale et al. 2010) allow users to efficiently apply standard machine learning algorithms to their work, without having to implement the algorithms themselves. Furthermore, these tools enable users who are not machine learning experts to effectively configure, train, and evaluate machine learning systems without having low-level knowledge of how the algorithms work.

Weka is a popular tool for applying supervised learning to many problem domains[2], and it supports a relatively comprehensive set of user actions, in comparison to other tools. For example, Weka provides a suite of GUI applications for performing open-ended explorations using supervised learning algorithms and for conducting experiments evaluating algorithms' performance. In Weka's "Explorer" and "KnowledgeFlow" interfaces, users can explore the application of different algorithms to different datasets: it is possible to load a dataset from a file or database, apply processing to the dataset (e.g., to normalize or rescale feature values), configure a learning algorithm (e.g., choose a classifier and set its parameters), and experimentally apply the algorithm to the data. Tools for this experimentation include visualizations of how the trained model classifies the training dataset, and an interface for computing cross-validation accuracy or test set accuracy. In the "Experimenter" interface, the user is able to design and execute suites of tests, for example to compare the cross-validation performance of several different algorithms on the same dataset and test for statistically significant differences in algorithm behaviors.

Weka also offers a Java API to allow users to use supervised learning algorithm implementations within their own code. It is possible for a user to perform all the aforementioned functions through calls to the Weka API. Developers may also embed trained models in their applications, so that the models may be used by their software (e.g., to classify users' incoming email as spam or not spam). Developers can even modify the source code of Weka's algorithms themselves, and they may integrate their own custom-designed learning algorithms into Weka by implementing them as Java classes compatible with Weka's object-oriented API.

RapidMiner offers much of the same functionality as Weka: it provides implementations of many standard learning algorithms, methods for visualizing datasets and model functions, and mechanisms to evaluate the performance of algorithms on datasets. Users may employ its GUI or integrate its engine into their own application code.

Libraries and frameworks such as libsvm offer narrower functionality. They provide standard implementations of learning algorithms that can be embedded into the user's code; but it is typically up to the user to write code to load a dataset, set the parameters of the learning algorithm, handle training and evaluation, and so on.

[2]As of 7 November 2010, Google Scholar indicates that the Weka book by Witten and Frank (2005) has been cited 11,437 times.

These tools make few assumptions about the sorts of applications to which they will be applied. That is, they are designed to be as general-purpose as possible; many of the constraints placed on the application of these tools arise from constraints embedded in the supervised learning algorithms themselves. For example, most algorithms cannot meaningfully take advantage of hierarchical relationships between features or classes, and some algorithms cannot use training examples that are missing one or more feature values. However, there are a few assumptions embedded in these tools—in particular, Weka and RapidMiner—that, as we will argue, do impose certain limitations on their applicability to certain problem domains: they assume that the training data comes from a database or file, that the set of data is more or less fixed in advance, that the user will not edit the training data (other than, for example, to perform normalization or other processing of the given examples), and that the user will evaluate trained models using measures of generalization performance estimated from the available data (for example, using cross-validation or test accuracy, or visualization of the model behavior on the data).

2.2.3 Supervised Learning in Music

In Section 2.1.2, we discussed the need for composers of interactive computer music to deal effectively with complexity in constructing the relationship between sensed performer actions and the computer's response, and/or in bridging the "semantic gap" between the sensed performer actions and the computer's interpretation of those actions. In the first case, the relationship between performer action and computer response in a composed interactive system can be defined as a model function that, given a set of inputs characterizing the performer's actions, produces a set of outputs that drive parameters of the computer's behavior (e.g., parameters of a synthesis algorithm). In the second case, in order to allow the computer to appropriately interpret a performer's actions, it is often helpful to create a model of the relationship between the sensor and/or audio signals generated by the performer and the semantic or musical meaning of the performer's actions (e.g., the identity of the gesture she has performed, or the tempo at which she is playing). Such a model allows the computer to be programmed to take action based on the higher-level interpretations of these actions, rather than based on the low-level signals generated by the performer.

In both these situations, the composer can often easily make data available, so supervised learning can be an effective tool for using data to create these models efficiently, and to create more accurate models than by other means. In this section, we provide an overview of three specific interactive computer music problems in which supervised learning has been commonly applied: the "mapping problem," gesture classification, and semantic audio analysis.

Creation of Mappings from Gesture to Sound

As discussed above, in interactive computer music systems, the computer may often function as an instrument (or one component thereof). Such a system is sometimes called a "digital musical instrument," which is defined by Miranda and Wanderley

(2006) as "an instrument that uses computer-generated sound..., [consisting] of a control surface or gestural controller, which drives the musical parameters of a sound synthesizer in real time." Such instruments allow a performer to gesturally control the computer's sound in an expressive, real-time manner akin to performing on an acoustic instrument. Historically, many digital synthesizers have been designed to to be played with a piano-like keyboard, such as a MIDI keyboard. While such interfaces leverage the existing skills of keyboard musicians, they offer limited dimensions of gestural control (i.e., choice of key, time of key press and release, and velocity of key press). These limitations often make keyboards a poor fit for creating digital instruments that do not mimic a keyboard instrument; for example, a standard keyboard does not afford performance techniques in which a player continuously controls sound synthesis parameters over time.

Therefore, musicians and researchers have increasingly chosen to employ other types of control interfaces that allow them to expressively perform digital sounds using a wide variety of control gestures. Performers might "play" digital synthesis algorithms with gestures sensed by sensors attached to the body (e.g., in work by Tanaka 2000 and Knapp and Tanaka 2002) or embedded in control interfaces (e.g., the Reactable by Jordà et al. 2007 or the SqueezeVoxen by Cook 2005), game controllers (e.g., in work by Steiner 2005), computer vision or motion tracking (e.g., Dobrian and Bevilacqua 2003), or native laptop inputs such as finger position on a trackpad (e.g., Fiebrink et al. 2007). Whereas physics dictates the relationship between performer gesture and sound in the performance of acoustic musical instruments, the creator of a digital musical instrument is quite free to design this relationship in new ways. Hunt et al. (2002) argue that this **mapping** from sensed input gestures to sound parameters can define "the very essence of an instrument."

Hunt and Wanderley (2002) have characterized two general strategies by which composers and instrument builders have designed mappings for digital musical instruments. In an **explicit** mapping strategy, the instrument designer explicitly specifies the mapping function from input sensed gesture to output sound parameters. This is typically done through coding or by designing functions in a graphical programming interface such as Max/MSP. In contrast, in a **generative** mapping strategy, the mapping is generated automatically from a set of example gestures paired with example sound parameters[3].

Supervised learning algorithms may be used for generative mapping creation: given a training set of gestures and corresponding sound parameters, the training step produces a model capable of producing new synthesis parameter outputs from new gestural inputs. Significantly, the feature space and the synthesis parameter space in these problems can be quite large, with many continuous dimensions of features and parameters, so this space often cannot be exhaustively represented in the training set. Fortunately, using continuous regression algorithms such as neural networks, the mapping function is capable both of producing outputs for gestures not

[3]Note that this use of the word "generative" is entirely different from how it is employed in machine learning, where a distinction is made between "generative" and "discriminative" algorithms. For example, it is quite possible to use a *generative mapping strategy* to build a mapping using a *discriminative learning algorithm.*

seen in the training set, and of producing synthesis parameters or (combinations of synthesis parameters) not seen in the training set. Generative mappings can also be used to control discrete synthesis parameters, either through the use of discrete classifiers or through the quantization of continuous models' outputs. For example, if the instrument designer wishes to create an instrument that, like a piano, is capable of playing only pitches in the 12-tone equal-tempered scale, he could train a classifier with 12 classes (e.g., to control pitch in a given octave) or 88 classes (e.g., to control pitch within the same range as a piano). Alternatively, he could train a continuous neural network model and round its real-valued outputs to the frequency value corresponding to the nearest discrete pitch.

Neural networks have been used for creating mappings since the work of Lee et al. (1991). In that work, the authors used neural networks for several real-time control tasks, one of which involved the use of a MIDI keyboard to control the timbre of a synthesized sound. Another early use of neural networks to create mappings was the work of Fels and Hinton (1995) mentioned above. In one component of their system, a neural network was trained to produce a continuous mapping function from hand gesture to vowels spoken by a speech synthesis engine, where continuous hand motions resulted in gradually changing vowel outputs. In that same system, Fels and Hinton used another neural network as a discrete classifier, which assigned static consonant phonemes to particular hand gestures. A third network was trained to output the probability that the user's current hand configuration indicated a vowel, and its output determined how the outputs of the vowel and consonant networks would be combined to produce the control parameters sent to the speech synthesizer. In other work, Modler (2000) used neural networks to generate mappings for a sensor glove in interactive computer music performance.

Mapping design for digital musical instruments is an area of active research and discussion in computer music. Several researchers have focused on the development of new mapping functions, for example using matrix operations (Bevilacqua et al. 2005) or local interpolation (Bencina 2005). Other researchers have focused on how general properties of a mapping influence the controllability, expressivity, and other properties of digital musical instruments. (Typically, this research has focused on properties of mappings other than whether they were produced using a generative or explicit strategy.) One characteristic of mappings that has been shown to be particularly important is whether the relationship between the sensed gesture parameters and the sound synthesis parameters is *one-to-one* (each synthesis parameter is controlled by exactly one gesture parameter, and each gesture parameter controls exactly one synthesis parameter), *one-to-many* (each gestural parameter may control multiple synthesis parameters; also called "divergent"), *many-to-one* (each synthesis parameter may be controlled by multiple gesture parameters; also called "convergent") or *many-to-many* (a combination of the above) (Hunt and Wanderley 2002).

Another mapping characteristic of potential significance is the extent to which the integrality and separability of input device dimensions are matched to the integrality and separability of synthesis parameters. Vertegaal and Eaglestone (1996) provide a summary of the definition and importance of integrality and separability in HCI: "Garner (1974) showed that certain parameters of a task are perceived as

being integrally related to one another (the user sees these as a unified whole), while others are separably related (the user sees these as a collection of separate entities). Consequently, users manipulate certain parameters simultaneously (such as the x- and y-position of a graphical object) while others are manipulated separably (e.g., the colour and size of a rectangle)... For optimal performance, it is important that the control structure of the device correlates with the perceptual structure of the task (Jacob et al. 1994)." Based on this understanding of the perceptual structure of a control task, mappings in which integral control parameters control perceptually integral sonic parameters and separable control parameters control separable sonic parameters may offer advantages.

Notably, this set of past work has primarily focused on the aesthetic and practical consequences of mapping properties that are more or less independent from the question of whether the mappings were created using explicit or generative strategies. There exists significantly less work examining the consequences of generative versus explicit mapping strategies, on either the type of digital musical instruments they produce, or on the larger process of composition and instrument design. However, certain work suggests that there are interactive benefits to employing generative mapping strategies; Merrill and Paradiso (2005), for example, found that end users of new digital instruments preferred to have the ability to customize mappings themselves, and a generative mapping strategy facilitated this activity. One of our goals in the research presented in this thesis is therefore to provide further insight into the consequences of generative mapping strategies for both performance and composition.

Finally, it bears mention that not all composers and instrument designers adhere to the notion of a mapping as a useful compositional construct. For example, Chadabe (2002) is critical of the the inability of mappings to adequately capture complex and indeterministic relationships between a performer and computer. For one thing, the goal of deterministic mappings that are constructed to replicate the performance roles of conventional musical instruments may be "to make the performer powerful and keep the performer in complete control." This emphasis on performer power and control runs contrary to Chadabe's own definition of interaction, which involves mutual influence between the computer and performer. In Chapter 9, we will revisit these criticisms of mappings in a discussion grounded in our own work with composers and instrument builders.

Gesture Classification

The goal of gesture classification is, in the simplest case, to assign a discrete class label to each user gesture. For example, as discussed above, Allen et al. (2003) created an American Sign Language gesture classifier that assigns to each human hand gesture (measured using sensors on the hand) the label of the English letter corresponding to that gesture.

Gesture classification can be used in several ways in interactive computer music performance. A classifier may be trained to recognize natural gestures of an acoustic performer—for example, to recognize the beat patterns of a conductor, as in work by Brecht and Garnett (1995) and Wilson and Bobick (2000), or the bowing articulations

of a violinist, as in work by Young (2008) and Rasamimanana et al. (2005). Accurate classification of these gestures enables the computer to react to the performer's actions in a musically appropriate way, for example by playing its part according to the conductor's tempo. Classification of these gestures can also enable a performer to directly control or influence the computer using a natural gestural vocabulary; for example, a cellist might influence the computer's playing style through changing her bowing articulations.

Gesture classification can also be used to recognize new gestural vocabularies that are customized for a particular performer, instrument, or composition. For example, in Winkler's dance/theater production *Falling Up*, a computer vision system identified locations of dancers on a stage; when dancers entered particular locations, musical events were triggered in the computer (Winkler 2002). In the next chapter we present an example discussing how gesture classification might be used to build a gesture-controlled drum machine, in which a performer controls the drum machine sounds using hand gestures in front of the webcam.

Like systems for gesture mapping, systems for gesture classification may use a variety of technologies to sense performer gesture and transmit this information to the computer. These technologies include custom sensor systems, game controllers, vision and motion tracking systems, and native laptop inputs, as mentioned above. Gesture classification might also be performed on the audio signal generated by an acoustic performer; for example, Tindale (2004) constructed a system for classifying percussion gestures based on the resulting audio, and Wanderley (2002) demonstrated that analysis of the audio signal of a clarinet player could reveal information about ancillary performance gestures (e.g., angle of the clarinet relative to the floor). Audio can also be used to classify non-musical sound-producing gestures, as was done in recent work by Harrison and Hudson (2008), who classified gestures based on the sound made by drawing shapes onto a wall or other hard surface using a "scratchy" finger.

In some gesture classification contexts, such as the vision-based triggering system of Winkler (2002) mentioned previously, the relationship between gestural features and the identity or intended consequence of a gesture is simple enough that supervised learning is not necessary. However, in gesture classification applications across many different gestural input modalities, the relationship between a gesture's features and its class label may be significantly more complex and difficult to understand or programmatically define. In these circumstances, supervised learning can be a useful tool for building models of the relationship between a sensed gesture and its class label. This is particularly the case when it is straightforward to provide a learning algorithm with a training dataset consisting of example gestures paired with the correct class labels. Additionally, human gesture classification systems must also be robust to natural human variation in the execution of gestures, as well as variations in environmental conditions such as lighting levels or sensor calibrations. Supervised learning algorithms' goal of producing models that are capable of generalizing and that are robust to superfluous variations in input features are well-matched to this requirement; users can often expect a trained model to be robust to these effects if such variations are present in the training data.

In the simplest case, a gesture classifier classifies static gestures (e.g., hand or body positions) from a feature vector describing the state of a vision- or sensor-based system at a single point in time. It is also sometimes possible to classify dynamic gestures using feature vectors that capture temporal aspects of the gesture; for example, features extracted at some fixed number of successive time points might be concatenated into a single vector, or the features themselves may describe temporal behaviors such as the number of local maxima within an analysis window, as was done by Harrison and Hudson (2008).

Alternatively, gesture *tracking* systems may involve more nuanced analysis than gesture classification. For example, they may analyze when certain gestures start and stop, and provide real-time analysis of how much of a gesture has been completed. Supervised learning algorithms can also be useful for gesture tracking, but this can involve a more sophisticated type of model capable of considering the temporal evolution of features and classes, such as Hidden Markov Models (Bishop 2007, Chapter 13).

Semantic Audio Analysis

Lastly, supervised learning has been well-established as a tool for the semantic analysis of audio signals. As humans, we readily perceive musical qualities such as tempo, pitch, harmony, and genre when we listen to a piece of music. However, the problem of creating a computer system capable of characterizing a piece of musical audio in these ways can be quite challenging.

Much research on computer analysis of musical audio is performed in the domain of music information retrieval, or MIR. Though MIR also encompasses much research on symbolic, textual, and other non-audio data, MIR research and the ISMIR conference in particular[4] maintain a significant focus on the semantic analysis of audio. The MIREX benchmarking competition, for example, includes audio analysis tasks for note onset identification, chord estimation, pitch estimation, tempo estimation, key detection, melody extraction, and beat tracking, among others.

Past research in MIR has identified a set of audio features and shown them to be relevant to certain semantic analysis tasks. These features include temporal measurements (e.g., the zero-crossing rate), spectral measurements (e.g., FFT bin magnitudes, spectral centroid, and spectral flux; see Tzanetakis et al. 2001b), wavelet-based features (Tzanetakis et al. 2001a), and other perceptually- and physiologically-motivated features, such as Mel Frequency Cepstral Coefficients (Logan 2000).

Though these features are often correlated with perceptually salient properties of the sound, the relationship between these features and higher-level musical properties such as harmony, tempo, or instrumentation can be exceedingly complex. State-of-the-art systems for identifying musical properties of audio therefore often use supervised learning approaches (see, e.g., systems evaluated in recent MIREX competitions[5]). Many MIR analysis tasks of interest involve classifying audio into discrete categories (e.g., the genre of an MP3 or the harmony of a short audio segment), and

[4] `http://www.ismir.net`

[5] `http://www.music-ir.org/mirex/wiki/MIREX_HOME`

researchers have commonly applied standard classification algorithms such as support vector machines (Poliner and Ellis 2005) or AdaBoost (Bergstra et al. 2006) to these tasks.

The majority of MIR research, and the evaluation methods employed in assessing the quality of MIR algorithms by ISMIR paper authors and by the MIREX competition, emphasize offline (i.e., non-real-time) analysis tasks. Examples of typical goals of MIR semantic analysis research include making music recommendations based on analysis of a user's music collection (e.g., Cano et al. 2005; Yoshii et al. 2006), annotating a personal or commercial music collection with semantically-relevant tags (e.g., Turnbull et al. 2008; Eck et al. 2007), or enabling a user to query a music database by humming a melody (e.g., Ghias et al. 1995; Dannenberg et al. 2004).

However, MIR research has occasionally focused explicitly on real-time audio analysis for the purposes of performance. For example, Murata et al. (2008) constructed a robot that adapted its singing based on the tempo of music in its environment, and previously-mentioned work by Raphael (2010) has created a computer system capable of expressively accompanying a human performer in real-time. Furthermore, many research systems for instrument identification, pitch classification, beat tracking, and numerous other tasks could be adapted from analysis of whole audio files to work in real-time performance, so long as (1) the audio feature extractors and analysis algorithms they employ can compute in real-time or faster, and (2) the feature extractors and analysis algorithms can compute on analysis windows of a "reasonable" length (perhaps a few seconds or longer, depending on the application).

While semantic audio analysis is not explored in depth in this thesis, the exciting potential for real-time audio analysis systems to be used more widely and more successfully in interactive computer music performance was a strong motivating factor of this work, and it is an application area we intend to explore further in the future.

2.3 Software Tools for Applying Supervised Learning in Music

Application of supervised learning to work in music has employed a mixture of general-purpose tools, music-specific tools, and custom software systems developed from the ground up by researchers and practitioners. In this section, we provide an overview of existing tools that people have used. Then, we discuss in detail our motivation for the creation of a new software tool that is general purpose, runs in real-time, is usable via a graphical interface, and supports a more interactive approach to machine learning systems-building, in order to better meet the needs of composers and musicians desiring to apply supervised learning to their work.

2.3.1 Existing Tools

The general-purpose tools discussed above in Section 2.2.2, such as Weka and Matlab, are often used to apply supervised learning to work in music information retrieval, where there is an emphasis on offline audio analysis. Other tools have been developed

to perform feature extraction and supervised learning tasks customized to the needs of MIR researchers, such as the jMIR Java toolkit (McKay and Fujinaga 2009), and the MIR Toolbox (Lartillot and Toiviainen 2007) and MIDI Toolbox (Eerola and Toiviainen 2004) tools that provide Matlab tools.

Many MIR tools—including GUI-based, general-purpose tools such as Weka and RapidMiner, and most tools such as jMIR that are customized to offline music analysis—are not suitable for application in interactive computer music, because they are incapable of being applied to real-time analysis problems. These tools assume that all relevant data exists in a file or database and typically are incapable producing a stream of output classifications in response to a stream of input features, for example.

It is still possible, of course, to use the Weka library and other libraries or frameworks to develop custom code for analyzing real-time signals. For example, one could develop the software infrastructure for extracting features, passing a stream of feature vectors to a trained classifier over time, and passing the stream of classifier outputs to control a musical process. Development of this infrastructure can involve a significantly greater time commitment than applying these tools to offline datasets, however, and this approach is only suitable for computer musicians who are also software programmers.

Marsyas (Tzanetakis and Cook 2000) is a software tool designed particularly for music information retrieval, and it is capable of analyzing both offline and real-time audio. Neither a programming language nor a GUI, Marsyas is a framework that provides a set of command-line tools and text-based and graphical mechanisms to define dataflow-based analysis patches composed of built-in and user-defined analysis building blocks. Users can develop larger Marsyas-based projects in C++ or using bindings to other standard programming languages, including Python, Ruby, Lua, and Java (Percival and Tzanetakis 2006). Marsyas enables trained classifiers (k-nearest neighbor, support vector machines, and Gaussian mixture models) to be packaged as plug-ins and run on real-time audio. It also includes some support for audio synthesis (wavetable synthesis, FM synthesis, and phase vocoder, in addition to file playback), although the motivation for adding this capability was not necessarily for use in for real-time music performance, but for use in development and debugging (e.g., allowing the developer to listen to extracted chords) (Tzanetakis and Lemstrom 2007). MarsyasX is a recent extension of Marsyas that can be applied to analysis of multimodal signals, including video and sensor data (Teixeira et al. 2008).

Composers and researchers have built several other tools that are specifically designed to apply supervised learning to real-time computer music performance. These tools are far less general-purpose than those described above, as they typically support only single learning algorithms, and/or particular programming environments and input modalities. For example, MnM (Bevilacqua et al. 2005) is a toolkit that allows users to create custom gesture-to-sound mappings using singular value decomposition, principle components analysis, hidden Markov models, and other algorithms. The toolkit is implemented as a suite of Max/MSP externals (i.e., processing objects that are used within Max/MSP's graphical patching environment). Cont et al. (2004) have developed a neural network-based gesture mapping toolkit for use in PD, an open-source graphical patching environment for music that is very similar

to Max/MSP. Fiebrink et al. (2008) created the "Small Music Information Retrieval toolKit" (SMIRK) for applying supervised learning algorithms (k-nearest neighbor, AdaBoost, and decision trees) to audio and gesture classification within the ChucK programming language (Wang and Cook 2003).

The software described above constitutes the state of the art in tools for applying supervised learning for music. We motivate our own set of requirements for a supervised learning tool for interactive computer music composition and performance in the next section, and we describe the tool we built to meet those requirements in the next chapter. Table 3.2 in the next chapter (page 3.2) provides a side-by-side comparison of the above tools with our new software, compared against our requirements.

Finally, we stress that not all music researchers and system builders have used the tools above in their work with supervised learning. The supervised learning components of several significant projects, such as the highly-specialized computer accompaniment systems created by Raphael (2010) and early, seminal work by Lee et al. (1991), have been developed entirely by the researchers themselves when there did not exist general-purpose tools or libraries that could be applied to their work.

2.3.2 Motivation and Requirements for a New Software Tool

The array of prior work applying supervised learning to music gesture and audio analysis attests to the capability for these algorithms to be effective in musical applications, especially when researchers and musicians have access to the tools they need to do their work, or when they possess the time and abilities necessary to develop such tools themselves. Unfortunately, there are many established and potential uses of supervised learning in interactive computer music for which the set of available tools is insufficient. Because of the requirement that that models run in real-time, the set of tools from which computer music composers and musicians may choose is much smaller than the set of tools available to many other applied machine learning researchers, or even researchers working in offline audio analysis. Furthermore, the tools that do exist for use in real-time computer music place significant restrictions on the types of algorithms that users may employ, the environments in which they may work (e.g., only Max/MSP, or only text-based programming languages), the tasks to which they may be applied (e.g., only to gesture mapping), and/or the degree of programming expertise and available development time that users must possess.

We believe that these constraints presented by current tools unnecessarily limit the pool of potential users of supervised learning (e.g., to only those who are proficient programmers, and to those who work in certain composition software environments), and they present unnecessary barriers to rapid prototyping and exploration (e.g., by requiring users to develop custom code to support even casual experimentation, and by forcing users to work in programming languages or composition software that are unfamiliar to them). As a result, these constraints may ultimately discourage the use of supervised learning algorithms by new types of composers and discourage their application to new types of musical problems.

Ideally, musicians and researchers would have access to a supervised learning tool that is general-purpose (i.e., that supports the application of arbitrary supervised

learning algorithms to arbitrary learning problems), that is capable of running on real-time signals, that is compatible with a variety of other software tools used in music composition and performance, and that does not require users to write code. Such a tool could facilitate experimentation with supervised learning algorithms by a greater number of composers applying them to wider array of problems. Such a tool could also lower the barriers to adapting existing supervised learning research conducted in offline or non-musical domains to live music. For example, as mentioned above, many research problems under investigation in MIR are potentially relevant and useful to computer music composers and performers, but there is often no easy way of adapting MIR research systems to run in real-time. Having a way to easily build an instrument classifier, for example, that uses the same features and learning algorithm as state-of-the-art systems in offline MIR research would enable composers apply this research to creative ends. Such a tool could also be useful for leveraging research in real-time contexts outside of music, for example work in computer vision-based gesture analysis, in the creation of interactive computer music systems.

In addition to being general-purpose, capable of running on real-time inputs, and GUI-based, we believe that a useful supervised learning tool for computer music composition and performance should also meet a set of more subtle requirements. First of all, conventional machine learning tools (and many music-specific tools described in Section 2.3.1) assume that the training data is present in a file or database. In music composition and performance, this might sometimes be the case: for example, in order to build an instrument classifier, one might cull training data from existing repositories of instrument recordings, such as the McGill University Master Samples library (Opolko and Wapnick 1989). On the other hand, in many problems in which a composer might wish to construct a supervised learning system, there exists no standard available training set. Furthermore, creating a custom, new dataset offers the benefits of enabling a composer to build a model that is customized to a particular performer, composition, or venue. For example, a composer might train a mapping function that allows a performer to play a specific synthesis patch using certain gestures, where the gestures and resulting sounds have been carefully designed by the composer to fit into a particular composition. Or, a composer might train a vision-based gesture classifier using a training set consisting only of gestures demonstrated by the musician who will be performing the piece, under the same lighting conditions that will be used in performance. In this case, the composer can take advantage of the training data creation process to not only customize the learning problem, but also to greatly simply the learning problem in comparison to the general problem of constructing a gesture classifier that is accurate for *all* performers and *all* lighting conditions.

Because of the likelihood that a composer or performer will want to create a custom training data set, it makes sense for a supervised learning tool to provide appropriate user interfaces for dataset creation. Furthermore, because the goal of the learning problem may often be to create a model whose inputs will be gestures or audio generated by a performer in real-time, it is appropriate that such user interfaces allow the training data to be generated by a user's actions in real-time. Some existing tools for real-time gesture analysis, such as examples distributed with the MnM Toolkit

(Bevilacqua et al. 2005), do incorporate support for users creating the dataset through real-time gesture demonstration. The SMIRK tool (Fiebrink et al. 2008), a smaller-scale project than this thesis that was nonetheless motivated by many of the same ideas, explicitly supports the interactive creation of both audio and gestural training sets. Other existing musical tools, such as the PD gesture toolkit by Cont et al. (2004), demand that the dataset be created in some other environment and saved to a file, and general-purpose tools such as Weka do not provide any support for this action.

Also, notably, each composer or musician may have different criteria defining what constitutes a "good" or "useful" model for a particular composition, instrument, or other system incorporating supervised learning. A composer might require a gesture classifier to produce very accurate classifications for the gestures of his performers, or he might only require that any likely classifier mistake result in musically acceptable consequences (e.g., sounds that don't sound "too bad"). An instrument designer might require that his created mapping produce sound output very similar to the sounds included in the training set, especially if he has a set of sounds that are important to play in a composition. On the other hand, it may be most important to him to build a mapping that his performers can learn to play in a musically sensitive and expressive manner, regardless of whether the instrument produces sounds that are anything like those in the training set. In any case, musical users' goals for supervised learning models may be more nuanced, more subjective, and possibly even unrelated to the conventional goal of supervised learning, which is to create models with high generalization accuracy.

Metrics such as cross-validation may or may not be informative of the subjective quality of a trained model; a supervised learning tool for musical use should therefore provide users with the means to evaluate trained models against their own criteria. In applications where the goal of using supervised learning is to create a model that produces outputs in response to performer-generated inputs, this can be accomplished by providing the user with an interface to evaluate trained models by running the models on input signals generated by himself in real-time. For example, a composer building a gesture classifier can evaluate the trained classifier by demonstrating new gestures in real-time and observing the classifier's output. This enables the composer to evaluate the classifier's accuracy on gestures most likely to be used in future performance, and—if the classifier's outputs are connected to the musical processes that it will be controlling during performance—this allows the composer to evaluate the musical consequences of the classifier's errors.

Finally, a system for applying supervised learning to interactive computer music should enable the user to take appropriate actions to improve a model, using information gained through evaluation. One important capability of general-purpose supervised learning tools such as Weka is that they enable users to experiment with different feature sets, learning algorithms, and parameterizations of those learning algorithms to build a model that best captures the relationship between features and labels present in the training dataset. Users who discover that a model they have built performs poorly can attempt to find an algorithm or feature subset that that produces a model that performs better. Because different learning algorithms model

the training data in different ways, and because different feature subsets may allow the data to be modeled more or less accurately, it remains appropriate to allow musical users of a supervised learning tool to improve their models through changing the learning algorithms and feature selection.

However, changing the learning algorithms or features may not always be the best method of improving a model in an interactive computer music context. Though changing the algorithm or features can allow the user to build a more accurate or faithful model of the relationship between inputs and outputs in the training dataset, the user might instead require a model that learns a qualitatively different type of relationship. For example, while evaluating a trained vision-based gesture classifier by demonstrating new gestures in real-time, the user may notice that certain variations of a gesture (e.g., changes in distance between himself and the camera) often cause that gesture to be misclassified. Or, while evaluating a trained gesture mapping, he may discover that the sounds produced by certain gestures he'd like to use in performance are, unfortunately, inappropriate to the composition he is composing. In both cases, changing the training dataset by adding new gestural examples paired with better labels may be a much more direct and effective way of improving the trained model, compared with using a different algorithm to model the original dataset.

Weka and other conventional supervised learning tools (including most musical tools mentioned above) do not explicitly provide support for the user to edit the training data, because the user's goal in applying supervised learning is assumed to be the creation of an accurate model of a given (and fixed) dataset. Changing this data simply to make the model more accurate would be counterproductive to that goal. For example, in the medical analysis work of Ross et al. (2003) mentioned above, if the researchers had edited the dataset by changing class values or feature values, or by adding hand-crafted training examples, their findings that the features were useful in predicting the class values would have been meaningless. Also, the trained models' ability to generalize to data similar to that in the original dataset would be compromised, so their usefulness as a tool for diagnosing new patients would be limited. On the other hand, the primary goal of composers and musicians employing supervised learning algorithms may be to build models that are most useful to them in real-time, interactive performances: they may be seeking to build models that accurately classify gestures or sounds produced by a particular performer, or that produce musically expressive mappings. The true goals of supervised learning are not represented definitively in the dataset, but in the imagination of the composer, where they are defined and informed by her knowledge, imagination, and experiences. The training dataset is merely the "interface" by which users communicate their goals to the learning algorithm, so it is entirely appropriate that users have the ability to modify the data to more clearly represent their goals.

2.3.3 Summary of Requirements

Based on the above discussion, we summarize our requirements for a new supervised learning tool for interactive computer music as follows:

1. **General-purpose in nature:**
   (a) Capable of creating models for arbitrary signal domains, including audio, sensor, video, and other signals generated by performers
   (b) Capable of being applied to arbitrary interactive computer music problems, including gesture-to-sound mapping, gesture classification, semantic audio analysis, and others
   (c) Includes support for many learning algorithms for both classification and regression
2. **Supports real-time applications:** Trained models can produce a stream of output labels in response to a stream of input features.
3. **Compatible with the tools composers already use:** Composers can use it alongside Max/MSP, PD, ChucK, and user-implemented code in other languages, among others.
4. **Supports interaction via a GUI:** Users without programming expertise can apply supervised learning effectively, and all users are free to do rapid prototyping and experimentation without the need for writing code.
5. **Supports rich end-user interaction with the applied supervised learning process:**
   (a) Users can create training data through interactive, real-time demonstration.
   (b) Users can evaluate trained models through interactive, real-time experimentation.
   (c) Users can improve trained models by changing algorithms, algorithm parameters, and features.
   (d) Users can also improve trained models by editing and deleting training data, allowing them to cultivate a dataset that best represents their goals for the supervised learning models.

2.4 Machine Learning and Human-Computer Interaction

Much prior research on applications of supervised learning to music has focused on the development and application of new algorithms. In contrast, our interests in developing in enabling supervised learning to become a more usable tool—i.e., a more efficient, effective, and satisfying tool—for a wider variety of users and supervised learning applications. Our design requirements, implementation, and evaluation of this system thus employ perspectives, values, and techniques from the domain of HCI.

There exists an exciting, relevant thread of recent work in HCI, in which other researchers have been investigating questions of how to make supervised learning algorithms more usable and more accessible to people working in various domains. This work has focused on better enabling end users—who may or may not be knowledgable of machine learning—to train, evaluate, and improve supervised learning systems. Goals of that research include both improving user experiences with supervised learning and improving the quality of the trained models, and prior research has pursued these goals through proposing innovations in user interface design, offering new types of human interaction with standard supervised learning algorithms, and designing novel algorithmic approaches to supervised learning that are capable of leveraging human expertise in new ways.

2.4.1 User Interaction with Training Data

Early work in this research area was conducted by Fails and Olsen (2003), who proposed that a significant opportunity for improving supervised learning systems lay in enabling the user to evaluate a model, then edit its training dataset based on his or her expert judgments of how the model should improve. They termed this approach "interactive machine learning," in contrast to conventional systems for applying machine learning in which the user does not edit the training set. Fails and Olsen implemented this interactive approach in their Crayons system, which allowed users to improve an image pixel classification model by iteratively adding training data, training a classifier, evaluating the classifier, and repeating. Users added training data through an interface that allowed them to quickly "paint" class labels (e.g., "skin pixel" or "background pixel") on previously-unlabeled image pixels within a fixed set of available images, and they evaluated the trained model by visualizing the model's classifications over each pixel in those images.

This approach to engaging user interaction in editing a model's training dataset has subsequently been applied to end-user supervised learning systems in handwriting analysis, web image classification, document analysis, and sensor-based interaction design. While these systems differ in the algorithms used and in the ways that interfaces support users' data editing and evaluation activities, they share a common interactive workflow, illustrated in Figure 2.3.

Shilman et al. (2006) created one such interactive machine learning system, a handwriting recognition tool called CueTip, in which users had the ability to interactively correct errors made by the handwriting recognizer as they wrote. Each user-initiated error correction resulted in the recognizer updating its handwriting model.

In 2008, Fogarty et al. created a web image classification system called CueFlik. The goal of CueFlik is to support "end-user interactive concept learning," in which the user trains a computer model to accurately rank images according to a user-defined concept, such as "product photos" or "pictures of people." As a user searches the Web for images, CueFlik allows users to iteratively select images returned by a query and add them as positive or negative examples to a concept training set. Additionally, as the user searches for images, she is able to apply the trained concept models to

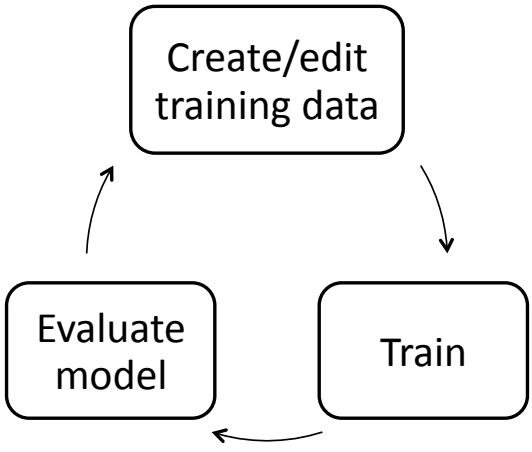


Figure 2.3: The interactive machine learning workflow used by Fails and Olsen (2003) and others.

re-rank images returned by the search; in this way, she is able to continually assess the performance of the model and take action to improve it through the addition of new training data.

Work by Amershi et al. (2009) extended the CueFlik system by implementing user interfaces for providing "overviews" of images associated with a concept. They demonstrated through a user study that, by providing users with relevant information about the models currently representing a concept, overviews could aid users in selecting better training examples to further improve the concept models. Work by Amershi et al. (2010) further extended this work by implementing a mechanism for allowing users to visualize the consequences of different potential modifications to the training dataset (i.e., of different potential image labelings) before committing to a modification. In this work, Amershi et al. demonstrated that enabling easier exploration and revision of concept models in this manner allowed users to build better models.

Baker et al. (2009) designed an interactive text document classification system in which users were able to add labels to documents to designate them as positive or negative examples of concepts, based on the contents of the documents. In that work, Baker et al. used a two-dimensional visualization of the document feature space to provide feedback to users about the extent to which documents of different classes overlapped. Their work suggests that users were able to employ these visualizations to understand the "inherent separability" of the dataset, which affects the extent to which any classification algorithm would be capable of learning the concept. Users were able to use this plot to assess the effects of modifications to the classifier algorithm, as well as to select documents based on their position in the feature space, examine them, and add new concept labels to them.

The Exemplar system created by Hartmann et al. (2007) enables users to design sensor-driven interactions using a "programming by demonstration" (PBD) paradigm.

In PBD, humans train computers and robotics systems to perform tasks by providing examples of those tasks; we further discuss PBD and its relationship to our approach to interactive machine learning in Section 3.6.3 (page 81). Exemplar users have the ability to provide examples of sensor manipulations (e.g., a firm press on a force-sensitive resistor), and to annotate sensor outputs with labels indicating how this type of manipulation should be acted on by the computer (e.g., to activate a light). Users can observe the pattern matching algorithm's output as they demonstrate subsequent sensor gestures in real-time, and they can employ a graphical interface to directly modify the matching algorithm to improve its behavior.

2.4.2 Other Opportunities for User Interaction

Other work at the intersection of HCI and machine learning has investigated approaches to leveraging other types of human interaction in the creation of supervised learning models. For example, the EnsembleMatrix system (Talbot et al. 2009) provides a graphical interface that harnesses human users to optimize ensembles of learners. The ManiMatrix (Kapoor et al. 2010) system allows users to manipulate a confusion matrix to interactively steer a model's performance to reflect their priorities.

Additionally, systems by Nichols et al. (2009) and Morris et al. (2008) provide users with interfaces for exercising control over the parameters of machine learning models. For example, Morris et al. (2008) present users with a "Happy Factor" slider whose effect is to control the relative weighting of two underlying models. By exposing algorithm parameters indirectly, and by visually presenting controls alongside labels that indicate how parameter changes are likely to affect characteristics of the system that are relevant to the user, these systems enable users who are machine learning novices to effectively customize machine learning systems according to their preferences.

The Gestalt system developed by Patel et al. (2010) takes another approach to supporting end-user application of machine learning. Gestalt is tailored to support software developers who are machine learning novices but who wish to incorporate trained classifiers into their software. It provides developers with visual tools, integrated into the IDE, for implementing and analyzing classification pipelines in their code.

2.4.3 Summary and Key Research Questions

This body of work demonstrates that the effectiveness of applied machine learning in practice rests not only on the quality of the machine learning algorithms and the quality of the data, but also on the user interfaces and interaction paradigms that machine learning practitioners employ to do their work. Research discussed in Section 2.4.1 has shown that, for domains where it is feasible for the user to generate training examples or apply labels to unlabeled examples, user creation and modification of the training dataset is an effective way to leverage human expertise in improving supervised learning systems. Additionally, research discussed in Sections 2.4.1 and 2.4.2 illustrates that appropriate user interfaces can enable even machine learning

novices to design working supervised learning systems, and it provides examples of how machine learning algorithms may be modified to take advantage of human input in new ways.

While this work has inspired our own endeavors to create usable supervised learning software for use in creating interactive computer music systems, there remain several key questions regarding interactive supervised learning that are pertinent to our research. These include questions about how to design user interfaces for training data creation, visualization, and editing within real-time applications; the nature and breadth of subjective criteria that may be important to users designing supervised learning systems for their own use, and how to provide appropriate interfaces for users to evaluate trained models against these criteria; how to provide feedback to users in order to enable them to interact most effectively with machine learning algorithms; and the variety of applications that different types of users—newly-empowered to employ supervised learning in their work—might imagine and create. We have explored these questions and others in our work observing and collaborating with musical users engaged with our new interactive supervised learning software, and we discuss our findings throughout Chapters 4 to 7.

2.5 HCI and Human Creativity

Another thread of relevant research in HCI considers how computer systems can effectively support human creativity, for example by enabling creative work to be more efficient or productive, or by enabling people to be creative in new ways.

Early work in this area by Shneiderman (2000) aims to ground the design of computer systems for creative work in an understanding of people's creative processes. He defines creative work broadly, encompassing not only work in the arts, but pursuits including the creation of knowledge and rapid communication among people. Activities such as letter writing, architecture, scientific research, and medical and legal practice are therefore considered to be creative endeavors.

Shneiderman outlines four phases of creativity in which technology can assist people to be "more creative more of the time": collecting information, relating (i.e., consulting with peers and mentors), *creating* (e.g., "explor[ing], compos[ing], evaluat[ing] possible solutions"), and donating (i.e., disseminating results). He also identifies eight activities that support creativity (and should therefore be the focus of further research in HCI): searching and browsing digital libraries, consulting with peers and mentors, visualizing data and processes, thinking by free associations, exploring solutions ("what-if tools"), composing artifacts and performances, reviewing and replaying session histories (to enable users to reflect on their work), and disseminating results.

Subsequent work by Shneiderman and others has led to the formulation of design guidelines for computer systems intended to be used in creative work (commonly called "creativity support systems" or "creativity support tools"). Guidelines proposed by Shneiderman (2007) and Resnick et al. (2005) include the need to "support exploration," "invent things that you would want to use yourself," and provide a "low

threshold, high ceiling, and wide walls" to accommodate both novices and experts, and to support a wide range of applications.

Much of this prior research is relevant to the design of software used in interactive computer music. For example, several of the creative activities outlined by Shneiderman (2000) are important to the work of computer music composers, instrument designers, and performers, and many of the design guidelines proposed by Resnick et al. (2005) were germane to the design of our new supervised learning software tool. In this thesis, we also present work that contributes to the larger body of computational creativity support research, in that we present new knowledge of users' priorities and values with regard to interacting with technology during computer music composition and instrument design, and in that we demonstrate how supervised learning algorithms can be particularly useful tools in creative work. We discuss these ideas in the context of particular user studies in Chapters 4 through 7, followed by a more detailed discussion in Chapter 9.

2.6 Conclusions

2.6.1 Summary of Foundational Prior Work

Interactive computer music is an active area of research and creative activity, and one that presents several interesting computational challenges related to the need to deal with complex relationships between human actions and computer reactions to or interpretations of those actions. Supervised learning offers an effective tool for working with complexity in domains where training data can be made available, and it has previously been applied to meet several of the computational challenges in computer music, including creating gesture-to-sound mappings in the design of new digital musical instruments and designing systems for human gesture classification and semantic audio analysis.

In many application domains, general-purpose tools such as Weka have enabled researchers who are not machine learning experts or programmers to effectively apply machine learning to their work. Tools such as jMIR (McKay and Fujinaga 2009), Marsyas (Tzanetakis and Cook 2000), and MnM (Bevilacqua et al. 2005) have been developed especially to aid researchers applying supervised learning to music, and they have had significant impact in particular applications in music information retrieval and music performance. However, there does not exist a general-purpose supervised learning tool that supports application of algorithms to arbitrary real-time problems, while also supporting appropriate user interactions with the supervised learning process.

A growing body of work at the intersection of HCI and machine learning has demonstrated the important role that human interaction can play in the end-user creation of supervised learning systems in other domains. In particular, enabling users to interactively and iteratively edit algorithms' training sets and evaluate trained models can be an effective and efficient way of engaging users' expertise to improve

supervised learning systems. Our work presented in this thesis has both been inspired by and contributed back to this body of research.

Finally, research on computational creativity support has underscored the roles that computers may play in enabling people to be more creative, and prior work has produced frameworks for understanding creative activities and proposed design guidelines for developers building creativity support tools. The work presented in this thesis builds on this foundational work in both the development of a new creativity support tool and the study of this tool in use by composers.

2.6.2 Statement of Research Motivations

Based on the success with which a variety of practitioners have been able to apply machine learning to problems inside and outside music when they have had access to software tools that enable them to work efficiently and effectively without requiring extensive machine learning expertise, we desired to create a new software tool whose general-purpose nature, real-time capabilities, and support for user interactions would enable more musical users to apply supervised learning more easily to more problems. Such a tool would have the potential to greatly benefit composers, musicians, and interactive computer music researchers—including ourselves—as well as practitioners working in domains with similar real-time and interaction requirements.

In Section 2.3.3, we outlined several criteria that such a tool should meet, based on the needs of composers that are currently unmet by existing tools. These criteria include the ability to apply arbitrary algorithms to arbitrary problems, to be compatible with a variety of software used by composers, to support accessibility to non-programmers and efficient exploration to all users by providing a graphical means of interaction, and to support interactions appropriate to composers' and musicians' needs. These interactions include creating the training data in real-time, modifying the training datasets, evaluating trained models by running them in real-time and manipulating their inputs in a hands-on manner, and iterating between evaluating models and improving them by making modifications to the learning problem.

We have built such a system, which we call the Wekinator. This software is described in detail in the next chapter. Some of the research subsequently presented in this thesis has been done with the motivation of improving the software using feedback from and collaboration with composers applying it to their work. Other research has demonstrated that the software in its current form accomplishes our goals of providing a useful, usable tool—one that has, in fact, been transformative for several of the composers who have used it.

Another significant component of our work with this system has also been to address a set of research questions impacting interactive computer music composition and performance, HCI, and machine learning. Through working with composers and musicians using our new interactive machine learning system, we have learned about the types of user interface support and interactions with machine learning that users have found most useful (and why). We have learned about the ways that both the system's support for human control over supervised learning and its feedback to users about the state of the learning process are instrumental in enabling

users to work effectively and efficiently, and in informing, educating, and inspiring users as they work. This research underscores the importance of interaction and interfaces in applied machine learning, and it demonstrates numerous examples of users benefitting from the ability to apply interactive machine learning to their work in the real world. Our research demonstrates new and effective interaction techniques that may be employed in interactive supervised learning in other real-time domains, and it shows that interaction can effectively "train" novice users to become more effective machine learning practitioners. This work also highlights important areas for future research that may better enable novice practitioners to apply machine learning effectively.

In this work, we have also acquired new knowledge of the human-computer interaction requirements of working composers and instrument designers. This research sheds new light on how the creativity support tool design guidelines and characterizations of creativity proposed in the HCI literature intersect with the values and goals of composers and instrument designers. Furthermore, we have demonstrated how supervised learning algorithms themselves can function as creativity support tools, for example providing access to rapid prototyping and exploration, higher-level thinking, and surprise and inspiration.

Chapter 3

The Wekinator: A General-Purpose Tool for Interactive Supervised Learning in Music and Real-Time Domains

3.1 Introduction

As discussed in the previous chapter, prior to this work, there did not exist an appropriate, general-purpose tool for interactively applying supervised learning to real-time musical problems, including audio and gesture analysis. Our requirements for a new tool, which we have outlined in Section 2.3.3, include compatibility with tools composers already use, interaction via a GUI, and the ability to interactively create training data, evaluate trained models, and modify algorithms, algorithm parameters, features, and the training dataset itself. In this chapter, we describe our new tool in detail, both to explain the novel approaches to interaction that it presents, and to aid the reader in understanding the user studies presented in the next four chapters, in which users applied our software to problems in music composition and instrument design.

When we set out to build our new tool for interactive supervised learning in music, we derived inspiration from the general-purpose machine learning and data mining toolkit called Weka (Hall et al. 2009). Weka has been successfully applied to problems in many application domains, including in our own work in offline audio analysis (e.g., Sinyor et al. 2005; Fiebrink and Fujinaga 2006; Fiebrink 2006). While Weka does not itself provide infrastructure for running trained models on real-time streams of input features, it can be used as a Java library inside other applications. Therefore, in order to speed development, incorporate standard and widely-tested implementations of supervised learning algorithms, and ensure maximal compatibility with Weka and other software using its APIs, we designed our new application to use the Weka library. As a tribute to the Weka project, and as an indication of our mission to

make our new software application even more powerful and useful in real-time music applications, we named our application "The Wekinator."

The Wekinator, first introduced in Fiebrink, Trueman, and Cook (2009), is a general-purpose software application for applying standard supervised learning algorithms to real-time problem domains. In addition to being capable of running trained supervised learning models on real-time input features to produce real-time outputs, the Wekinator provides user interfaces to allow real-time, interactive creation and editing of training datasets. One such interface introduces a novel, "playalong" mechanism for creation of training data, in which the Wekinator constructs training examples in real-time by extracting features from a user acting as if he were "playing along" or controlling a sequence of target model outputs. The Wekinator also provides support for an interactive style of supervised learning, allowing the user to perform all of the following actions within a single graphical user interface, and in any reasonable order: modify the training data and selected features, change the learning algorithms and their parameters, train and retrain models following these modifications, evaluate algorithms' cross-validation and training set accuracy scores, and run trained models in real-time on new inputs.

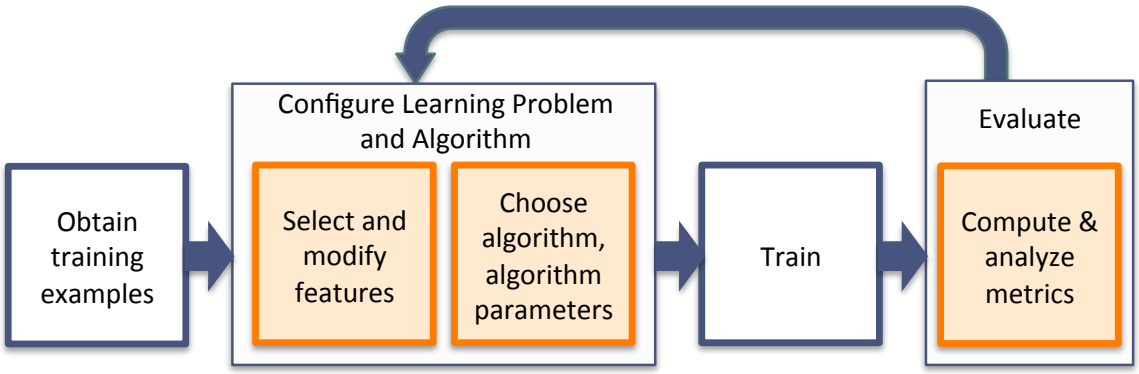
The Wekinator is capable of being applied to arbitrary real-time classification and regression problems, as it can receive input feature vectors from any type of source and send models' outputs to any destination in real-time. Aspects of its implementation are tailored, though, to facilitate its application to music. The Wekinator contains several built-in feature extractors for gestural and audio inputs, and it provides a simple way for using model outputs to control audio synthesis programs written in ChucK, a music programming language (Wang and Cook 2003).

In this chapter, we describe in further detail the Wekinator's support of interactive supervised learning, its architecture and implementation, and its user interface and support for user interaction. We also discuss how the approaches to interactive supervised learning supported by the Wekinator relate to other research and applications in domains including HCI, music, robotics, and speech.

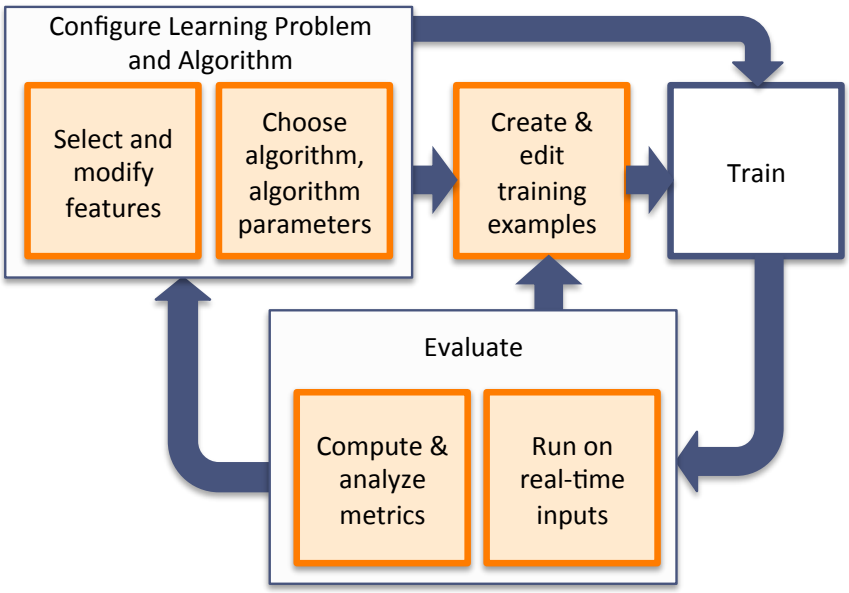
The description of the system found in this chapter is current as of the writing of this thesis. Many features of the Wekinator described here were implemented as a result of the user studies described in subsequent chapters, and we discuss such improvements in detail in those chapters. The Wekinator continues to undergo development, and readers are encouraged to visit the project homepage at `http://code.google.com/p/wekinator/w/list` to download the current release, join the mailing list, and view instructions for the Wekinator's installation and use.

3.2 Interactive Supervised Learning

In Section 2.4.1 in the previous chapter, we introduced the definition of interactive machine learning as proposed by Fails and Olsen (2003), and we discussed related work in the field of human-computer interaction. In this section, we provide a high-level discussion of the interactive supervised learning workflow supported by the Wekinator,



(a) A conventional supervised learning workflow.



(b) The Wekinator supervised learning workflow.

Figure 3.1: Workflows in conventional machine learning and in the Wekinator. In 3.1a, human expertise is engaged in selecting features, choosing the learning algorithm and its parameters, and analyzing evaluation outcomes; the features or algorithm may be modified based on the outcomes of evaluation. In 3.1b, human expertise is additionally engaged in creating and editing training data and running the model on new inputs in real-time; the features, algorithm, or training data may all be modified based on the outcomes of evaluation.

followed by an example illustrating how this workflow might be used to create a gesture-controlled drum machine.

3.2.1 Interactive Workflow

We begin by contrasting the interactive workflow supported by the Wekinator, shown in Figure 3.1b, with that supported by the Weka application and similar tools for non-interactive supervised learning, shown in Figure 3.1a. The workflow of Weka assumes that the learning problem is represented by a finite, fixed dataset, and that the goal of applying supervised learning is to model this dataset faithfully (though typically preserving concern for generalization accuracy, as discussed in Section 2.2.1). The user may interact with the system by modifying the feature set (e.g., selecting some features and ignoring others, normalizing features, projecting the dataset into a lower-dimensional space, etc.), changing the learning algorithm or its parameters, and evaluating an algorithm's performance by computing accuracy metrics, examining confusion matrices, or visualizing a classifier's decision boundaries.

In many problems for which the Wekinator was designed, it makes sense to allow the user to edit the training dataset as part of the process of creating a supervised learning model. In audio and gesture analysis problems, for example, it can be straightforward for a user to create training data by demonstrating gestures or sounds while providing their desired labels. Additionally, it makes sense to allow the user to evaluate trained models by running them on real-time inputs—including those generated by the user, through the same mechanism used for interactive training data creation—and to address model shortcomings identified during evaluation by modifying the training data, algorithm, algorithm parameters, and features. The Wekinator supports all these actions within its user interface.

3.2.2 Example: A Gesture-Controlled Drum Machine

We will now illustrate how the Wekinator might be used for interactive supervised learning in a real-time domain, using the example problem of creating a computer vision-based gesture classifier that allows a performer to "play" a digital drum machine in a live performance. To begin with, the user chooses a vision feature extractor to extract meaningful features from his gestures in front of a camera (possibly the webcam built into his laptop). This could be one of the Wekinator's built-in feature extractors or part of another software package, as we discuss later. Also, the user must initially write or obtain a drum machine application, in ChucK or some other environment. The number and type of parameters of this drum machine determine the number and type of models that will be created: a regression model will be created for each real-valued parameter, and a classifier model will be created for each discrete- or nominally-valued parameter. In the simplest case, the drum machine might take one discrete-valued parameter indicating how many drum loops it should play at once. For example, one loop might provide a steady bass beat, and another loop might add a faster hi-hat beat on top.

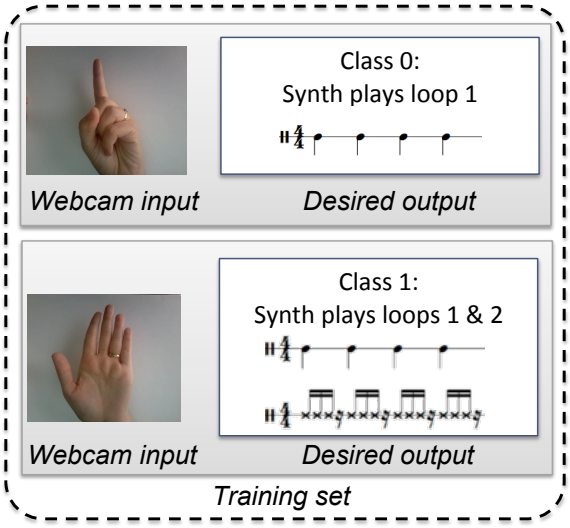


Figure 3.2: An example training set, consisting of two training examples matching a gesture with a class label and, by extension, a synthesizer's output.

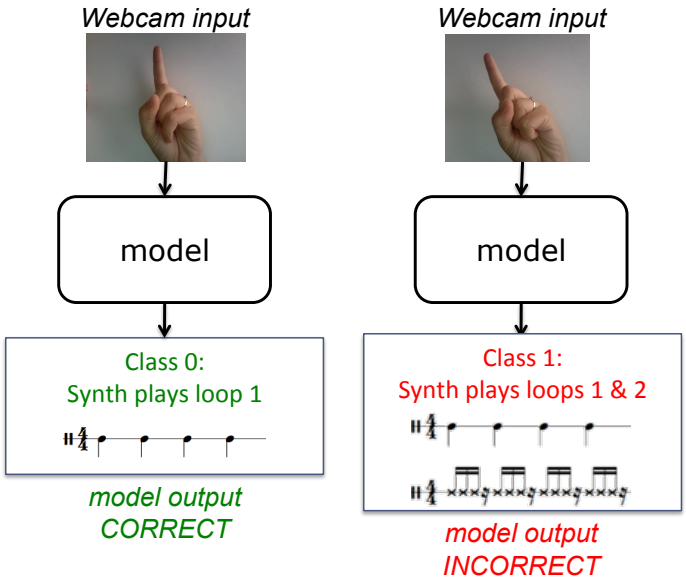


Figure 3.3: After training on the training data in Figure 3.2, the user may evaluate the model by running it on new gestures demonstrated in real-time.

Next, the user must create a training dataset. He first chooses some control gestures and the drum machine control parameter (and, by extension, the drum machine sound) that will correspond to each one. For each gesture, the user demonstrates the gesture to the Wekinator while using the GUI to indicate which drum machine parameter value should correspond to that gesture. Figure 3.2 illustrates a simple example training set created by the user. After at least one training example has been created, the classifier can be trained.

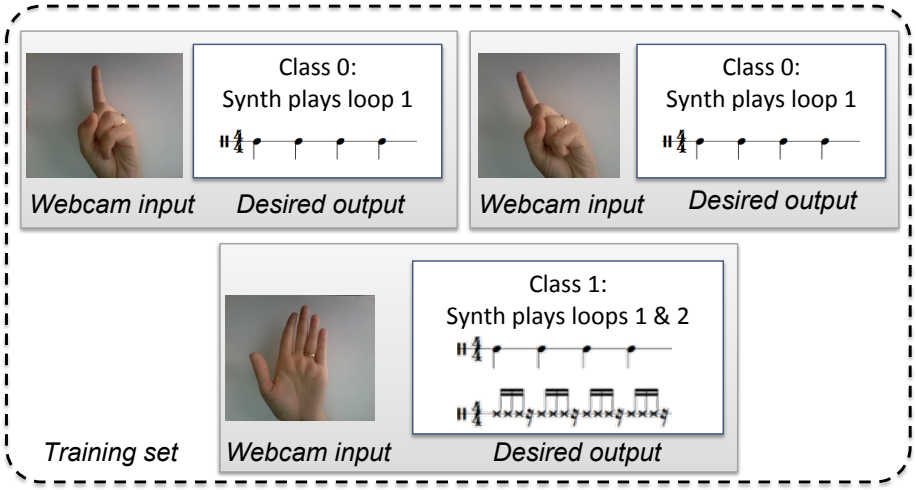


Figure 3.4: After the evaluation in Figure 3.3, the user may add training examples to correct mistakes.

The user can evaluate the trained classifier by performing gestures in front of the webcam and listening to how they affect the drum machine's sound (Figure 3.3). He may find that, for all gestures that he would want to use in a performance, the model performs perfectly; in that case, he can save the trained model to a file and run it during his next performance. Or, he may discover that, for certain gestures, the model produces incorrect classifications. At that point, the user may stop evaluating the model and attempt to fix the problem by modifying the training set. In particular, he may create additional training examples by demonstrating gestures similar to those classified incorrectly, but supplying the correct label for those gestures in the training data (Figure 3.4). The user can then retrain on this modified training set, evaluate the new model, and iteratively modify the training data and retrain until he has created a model he likes.

Once the trained model is performing well, the user might decide to make the classification problem harder, adding another gesture class to the training dataset and retraining (Figure 3.5). Or, he could choose to make the model more robust to changes in lighting conditions or to other performers' hands, by adding training examples with lighting and performer varied. Alternatively, the user might decide that he has a better idea for more interesting control gestures, and he can delete the training dataset and recreate it from scratch using these new gestures. In each case, he can use the dataset to represent to the computer his current definition of the learning problem, and he can count on the algorithm to try to faithfully model the problem from this data.

If the user has reason to believe that a different classification algorithm might produce a better model from the data, he can change the learning algorithm or its parameters at any time and retrain on the same dataset, producing a new model for him to evaluate. He can also try to improve the training dataset's representation of the learning problem by changing the features, for example specifying that the

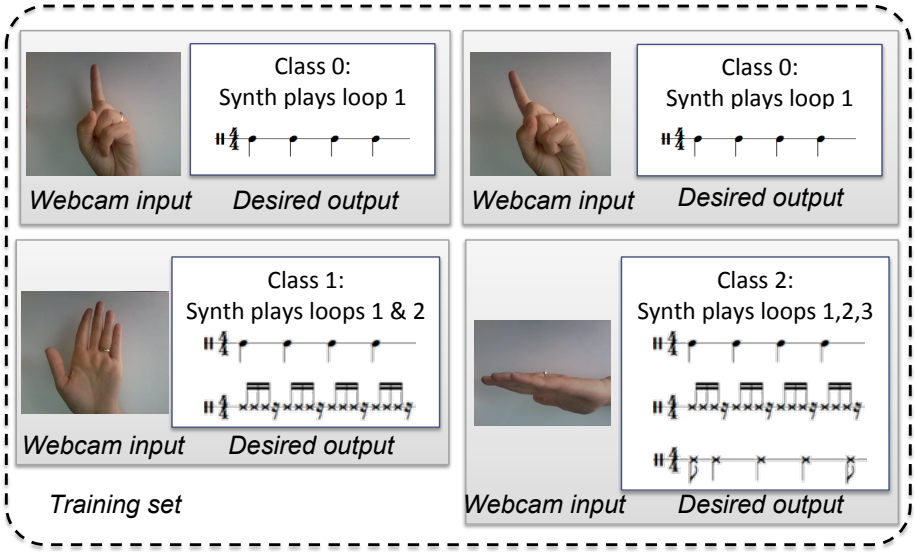


Figure 3.5: The user might also add training examples to make the problem harder, for example by adding a new gesture class.

algorithm should ignore features that are likely to be noisy or irrelevant, or adding new features that capture essential qualities of the input gesture.

Throughout the learning process, it is up to the user to evaluate whether he likes how the model is behaving, and to take any appropriate action to attempt to improve or change it. While it is possible to compute standard evaluation metrics to objectively assess how well an algorithm can model the current dataset, much of the user's assessment of a model may ultimately rest on how suitable it is for the task for which it is being designed; in this case, it may be most important that the model allow a live performer to play the drum machine in an easy, accurate, and musically expressive way. By running the current drum machine model in real-time on new gestures, the user can assess whether the model meets these criteria, and if not, form ideas about how he would like to change its behavior.

Notably, it is also possible that the user may change his evaluation criteria defining what a "correct" model entails. Perhaps he discovers during running a trained classifier does not do what he intended it to do, but instead does something he likes better. He might choose to keep the classifier as-is, or he may even decide to add more training data to reinforce the newly-discovered behavior. Or perhaps the user began his experimentation with the Wekinator without any clear ideas of what gestures he wanted to use in performance; in that case, his running of the model on real-time gestures is a means of not just assessing the model, but of evaluating the gestures themselves, to discover how they might "feel" in performance.

The user may take a variety of other actions during work with the Wekinator, in addition to the actions discussed above. For example, he may load and save trained models, visualize the training data, and compute objective evaluation metrics, and these actions are discussed in the detailed discussion of the software in the following sections.

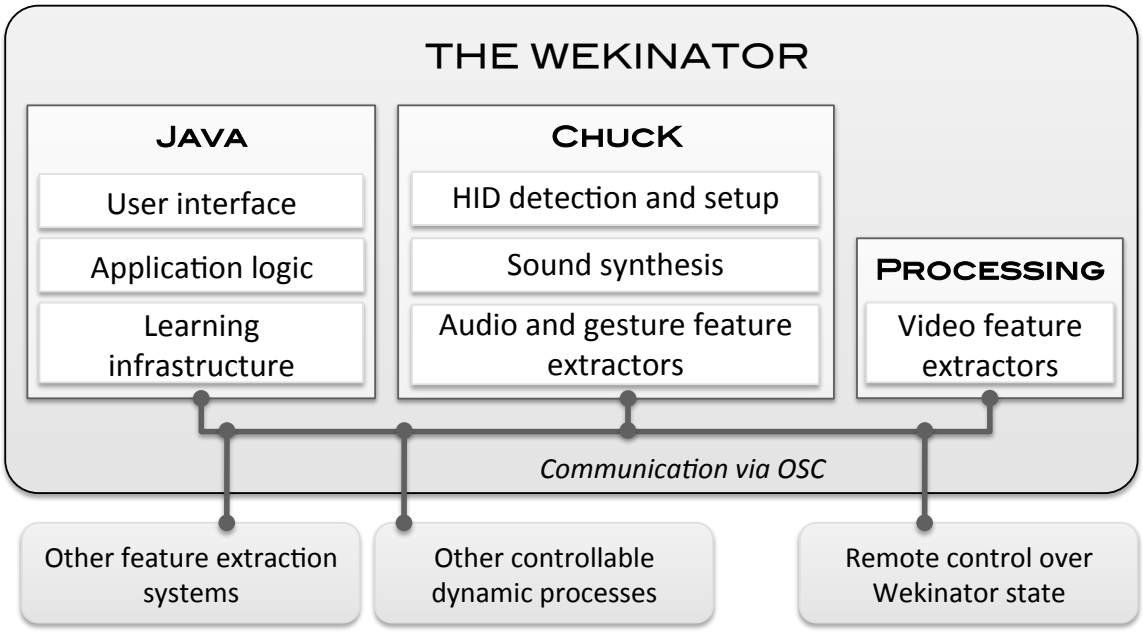


Figure 3.6: The basic Wekinator architecture. Components written in Java, ChucK, and Processing communicate via OSC. Optionally, external feature extraction systems and controllable processes may be used, and the Wekinator state may be remotely controlled. Communication between the Wekinator and these external processes also uses OSC.

3.3 System Overview

The Wekinator is a cross-platform tool written primarily in Java and ChucK. Figure 3.6 illustrates the roles of the different components that make up the Wekinator. The Java component of the system implements the user interface, application state and logic, and the learning infrastructure built on Weka. The ChucK component, which runs separately in its own virtual machine, contains modules for the detection and setup of HID devices, the control of sound synthesis algorithms written in ChucK, and both built-in and user-defined feature extractors for audio and gestural inputs. The Wekinator also comes with two basic computer vision feature extractors written in the Processing language (Reas and Fry 2007).

3.3.1 Open Sound Control

The ChucK, Java, and Processing components that comprise the Wekinator communicate via a lightweight, UDP-based protocol called Open Sound Control (OSC) (Wright and Freed 1997). OSC is also optionally used to communicate features from external feature extractors to the Wekinator, to communicate trained models' outputs to external applications for controlling sound synthesis or other dynamically

processes, and to control the Wekinator state itself from an external process, for example a user-created GUI.

OSC messages include an address (hostname and port), a URL-style message name, and a bundle of string and/or numeric data. Because OSC messages are sent over UDP, each communicating software component may optionally be run on different physical machines connected to the same network. Although the ChucK and Java components of the Wekinator will typically be run on the same machine, it is occasionally very useful to run feature extractors on a separate machine from the rest of the Wekinator for computational or debugging purposes. This was the practice in the bow gesture classification work discussed in Chapter 6, for example. Additionally, this enables input features, output parameters, and control messages to be communicated to and from other types of networked hardware devices, such as an iPhone running an OSC multi-touch interface application like TouchOSC[1] or a Make Controller[2].

OSC was developed for use in music in 1997 to address several shortcomings of the MIDI protocol for communication among musical hardware and software devices. Since then, it has been integrated into many music software packages and programming languages used by computer musicians and composers, as well as into many multimedia and other non-musical software systems. For example, most commonly-used music composition environments support OSC, including Max/MSP[3], Pure Data[4], SuperCollider (McCartney 2002), and ChucK (Wang and Cook 2003). OSC is also supported by commercial music products including Ableton Live[5] and Reaktor[6], and by graphics and video processing packages popular among multimedia artists, such as Processing (Reas and Fry 2007), Jitter[7], and Veejay[8]. Furthermore, OSC libraries exist for most standard programming languages, so a developer seeking to write a new feature extractor or Wekinator-controlled process can use the implementation language of his or her choice.

3.3.2 Feature Extraction

The Wekinator can receive input features from any of its built-in feature extractors, a ChucK feature extractor implementing a specified API, or an arbitrary external feature extractor capable of sending messages via OSC. In all cases, feature vectors are communicated to the Wekinator and represented within the Wekinator as a vector of floating-point values.

[1] `http://hexler.net/software/touchosc`
[2] `http://makezine.com/controller/`
[3] `http://cycling74.com/products/maxmspjitter/`
[4] `http://puredata.info/`
[5] `http://www.ableton.com/`
[6] `http://www.native-instruments.com/`
[7] `http://cycling74.com/products/maxmspjitter/`
[8] `http://www.veejayhq.net/`

Built-in Feature Extractors

The Wekinator's built-in feature extractors support the use of basic audio and gestural inputs. The built-in audio feature extractors use the ChucK Unit Analyzer framework for real-time signal processing, developed by Wang et al. (2007), and the available features include FFT magnitude spectra, spectral centroid, spectral flux, spectral rolloff, and RMS. These features are commonly used in audio analysis and music information retrieval, and they capture information about a sound's pitch, timbre, and energy (see, e.g., Tzanetakis et al. 2001b).

The built-in gesture feature extractors include support for gestural input sensing using a laptop's accelerometers, trackpad or mouse, and webcam, following work on the SMELT project by Fiebrink et al. (2007). For acceleration and tilt, the Wekinator uses Apple laptops' internal Sudden Motion Sensors; their primary purpose is to sense sudden changes in acceleration and "instantly [park] the hard drive heads to help reduce the risk of damage to the hard drive on impact" (Apple Inc. 2008), but their three axes of acceleration can also be polled in real-time from the Wekinator. For trackpad or mouse inputs, the Wekinator extracts two features indicating the x- and y-coordinates of the on-screen cursor, allowing the user to use the screen as a 2D position controller. For webcam inputs, the Wekinator is packaged with two basic vision feature extractors written in Processing, one that extracts 100 features from from a down-sampled edge detection process, and another that extracts 6 features pertaining to the absolute and relative position and angle of two tracked colored objects, in colors of the users choosing[9]. These vision extractors allow a user to control the Wekinator using, for example, hand gestures such as those in Section 3.2.2, larger-scale body movements, or movements of ad-hoc, colored fiducials (e.g., a colored pen or coffee cup).

The Wekinator also provides built-in support for gestural feature extraction from USB gaming devices that conform to the Human Interface Device (HID) protocol (USB Implementers' Forum 2001). While HID devices may take any physical form, they represent their physical states using a common set of descriptors. The Wekinator is able to extract features from a device that represents its state using one or more "axes" (controls that may be continuously varied), "buttons" (on-or-off controls), or "hat switches" (discrete controls with a finite number of selectable states, for example a direction pad). Common HID devices include joysticks (such as the one shown in Figure 3.7), gamepad controllers, and dance pads.

External ChucK and OSC-Enabled Feature Extractors

Users can write new feature extractors in ChucK by implementing the Wekinator's API for custom ChucK feature extractors. (Because ChucK's simple inheritance system does not formally allow the declaration and implementation of an interface, implementing the Wekinator's custom ChucK feature extractor API entails implementing a class whose name and method signatures are identical to those specified

[9]The edge detection extractor is adapted from code written by Tom Lieber, and the color tracker is adapted from code written by Nikolaus Gradwohl.

Figure 3.7: This Logitech joystick is an example of a HID device that musicians have used with the Wekinator.

in the Wekinator documentation.) A custom ChucK feature extractor computes the features as a vector of floating-point numbers and provides methods for the Wekinator to access their values and names. Figure 3.8 shows a ChucK code example implementing part of a custom ChucK feature extractor.

The Wekinator's support for custom ChucK feature extractors allows users familiar with ChucK to write customized audio analysis or gestural input feature extractors at a high logical level, without having to write their own code handling the communication of features to the Wekinator. Also, because of ChucK's cross-platform implementation, ChucK audio and HID code can be written without concern for the underlying operating system's approach to audio or HID device handling.

Users can also implement their own feature extractors in other environments or use other existing feature extraction applications, so long as these features are communicated to the Wekinator via OSC. This entails simply packaging each feature vector as a set of floating-point values along with a specified OSC message name string, then sending this message to the UDP host and port where the Wekinator is listening. External feature extractors may also communicate their feature names to the Wekinator over OSC for display in the Wekinator GUI.

Any number of features (up to limits imposed by OSC message size restrictions) from different sources can be used in the same supervised learning problem. Feature

```
public class CustomFeatureExtractor {
      //This extractor only computes 1 feature
      1 => int numFeats;

      //Extractor stores computed features in this array
      //Wekinator regularly grabs values directly out of this
      //array, according to its feature extraction rate
      new float[numFeats] @=> float features[];

      //Set the rate at which these features are computed
      100::ms => dur defaultRate => dur rate;

      //A simple analysis patch: Compute an FFT from audio input
      adc => FFT fft => blackhole;

      //Extraction loop: Extracts features at a rate of "rate"
      //Starting & stopping of this loop controlled by Wekinator
      fun void extract() {
            if (! isExtracting) {
              1 => isExtracting;
                  while (isExtracting) {
                        computeFeatures();
                        rate => now; //wait 100ms
                  }
          }
      }

      //When called, computes feature vector once
      fun void computeFeatures() {
            //trigger computation of FFT
            fft.upchuck();
            //My feature is magnitude of 0th FFT bin
            fft.fval(0) => features[0];
      }
      //... Other methods go here, including those called by
      //Wekinator to start and stop extracting; method for
      //communicating feature name(s) to Wekinator
      //...
```

Figure 3.8: Example ChucK code implementing part of a user-defined ChucK feature extractor class. This feature extractor computes the magnitude of the lowest-frequency bin of an FFT.

extraction may occur at any rate, and this rate may be fixed or variable. The rate at which built-in audio features are extracted can be set and changed in the Wekinator GUI, and built-in feature extractors for native laptop inputs are event-driven, causing a feature vector to be sent when a new action is performed. Due to restrictions imposed by the types of learning algorithms employed by the Wekinator, all feature vectors must be one-dimensional vectors of real numbers, and the length of the incoming feature vectors must remain consistent within the training data and the features to be classified by the trained models.

3.3.3 Learning Algorithms, Models, and Parameters

One Model Per Parameter

The Wekinator uses standard supervised learning algorithms to build one or more models, each of which is a function capable of computing a real-valued output in response to an incoming vector of real-valued features (see Section 2.2.1). As the Wekinator receives a new feature vector, each of its trained models computes its output value from this vector. In the typical Wekinator usage scenario, this set of model outputs is used to control the parameters of a dynamic process of some sort. For this reason, we will commonly refer to the outputs generated by the Wekinator as "parameters." These parameters might control a digital synthesis algorithm, for example causing changes in pitch, volume, or timbre. Or, these parameters might drive higher-level processes in an interactive computer music performance, for example triggering changes in section, tempo, or style. Alternatively, the parameters could control low- or high-level behaviors of interactive animations, robotic systems, video games, or other arbitrary processes.

The Wekinator will create one model to drive each parameter of the process being controlled. Currently, the Wekinator will create a neural network for each parameter that is real-valued and unbounded in range, and it will create a classifier for each parameter that is discrete or nominally-valued. The parameters need not be of the same type, and different algorithms may be used for each parameter. For example, Figure 3.9 shows how two neural networks, a support vector machine, and a k-nearest neighbor classifier might be used to drive four synthesis parameters.

It is up to the process using the parameters to decide how to interpret and use the Wekinator's outputs. The raw values output by the Wekinator may be used as-is, or the code receiving the parameters may threshold, scale, smooth, or otherwise post-process them before applying them for control. For example, for the learning problem in Figure 3.9, the synthesis code would likely limit the two neural network outputs to ensure that they conformed to the legal range of values for the PitchChangeRate and FilterQ parameters. Additionally, if the Wekinator's output parameters are used to trigger events as opposed to continuously controlling real-time behaviors of a process, the controlled process might include logic for determining when events should happen. For example, a process might trigger the playing of a note only when a discrete classifier's output class changes.

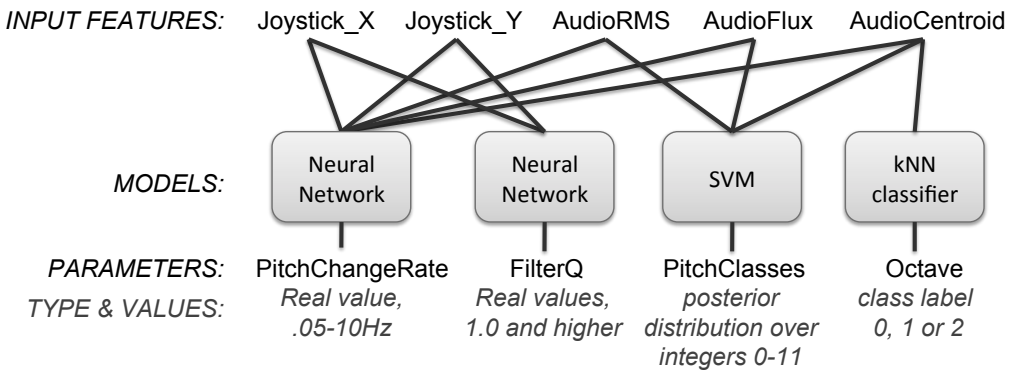


Figure 3.9: An example of a learning problem to which the Wekinator might be applied. In this example, four models drive four synthesis parameters based on the values of five input features. A different subset of features is selected for each parameter model, and the synthesis parameters have different types, different modeling algorithms, and different effects on the sound.

Selecting Features and Training Data for Each Model

The Wekinator allows the user to specify which of the input features will influence which of the output parameters, that is, which of the available features will be selected in the training and running of each model. This mapping from input features to output parameters can take any configuration, including one-to-one, one-to-many, many-to-one, and many-to-many. Figure 3.9 shows an example Wekinator configuration in which the PitchChangeRate parameter is influenced by all features, the FilterQ and PitchClasses parameters are influenced by subsets of features, and the Octave parameter is controlled by only one feature.

The Wekinator also provides the user with the ability to select *training examples* for each model. We further explain and discuss this functionality in Section 3.3.5.

Algorithms

In this section, we provide an overview of the learning algorithms used in the Wekinator. The reader is encouraged to consult textbooks by Bishop (2007) and Witten and Frank (2005) for further information about these algorithms and their implementation in Weka, respectively. These standard algorithms are all commonly used in problems within music analysis as well as other applied machine learning domains. Most have additional parameters beyond those described here, but we have omitted their discussion for simplicity. The set of learning algorithms used in the Wekinator, along with the algorithm parameters that the Wekinator exposes to the user, are summarized in Table 3.1.

The neural network algorithm used to build models for real-valued, continuous parameters is Weka's MultilayerPerceptron neural network, which is a feedforward neural network that uses error backpropagation for training (Witten and Frank 2005).

Table 3.1: The learning algorithms supported by the Wekinator, the underlying Weka class used for each algorithm, and the parameters the Wekinator exposes to the user.

Name	**Weka Class**	**Parameters**
Multilayer perceptron neural networks	MultilayerPerceptron	Architecture, # training epochs, learning rate, momentum
k-nearest neighbor (kNN)	IBk	k (# neighbors)
J48 decision tree	J48	none
AdaBoost.M1	AdaBoostM1	# training rounds, boost on decision trees or stumps
Support vector machine (SVM)	SMO	Complexity constant, kernel (linear, polynomial, RBF). For linear kernel: No extra parameters. For polynomial kernel: exponent, whether to use lower-order polynomial terms. For RBF kernel: gamma.

A multilayer perceptron network is comprised of a series of computational nodes, each of which produces an output based on a nonlinear activation function on its inputs (Bishop 2007, Chapter 5). These nodes are interconnected, forming a model comprised of multiple layers of logistic regression models, where the features are used as inputs to the first layer of nodes, and the model outputs are comprised of the outputs of the last layer of nodes. Neural networks are capable of modeling complex, non-linear relationships between input features and output values, and the trained network provides a compact representation of the model function, allowing for efficient computation on new inputs.

For a learning problem with f features, the Wekinator creates a neural network whose default architecture includes an input layer with f nodes, a hidden layer with f nodes, and a single output node. Pairs of nodes between the input layer and hidden layer, and between the hidden layer and output node, are fully connected. The default network for a learning problem with five features is shown in Figure 3.10. The user is able to change this architecture, as well as certain parameters of the backpropagation training algorithm, using the GUI described in Section 3.4.5 and shown in Figure 3.28.

A classification algorithm is used to build a model for each discrete-valued parameter, and the number of discrete classes is determined by the range of legal parameter values. For each discrete parameter model, the Wekinator can output either the single best class label or an estimated posterior distribution over all labels. (As none of the classifiers implement probabilistic models, this posterior is produced using heuristic methods.)

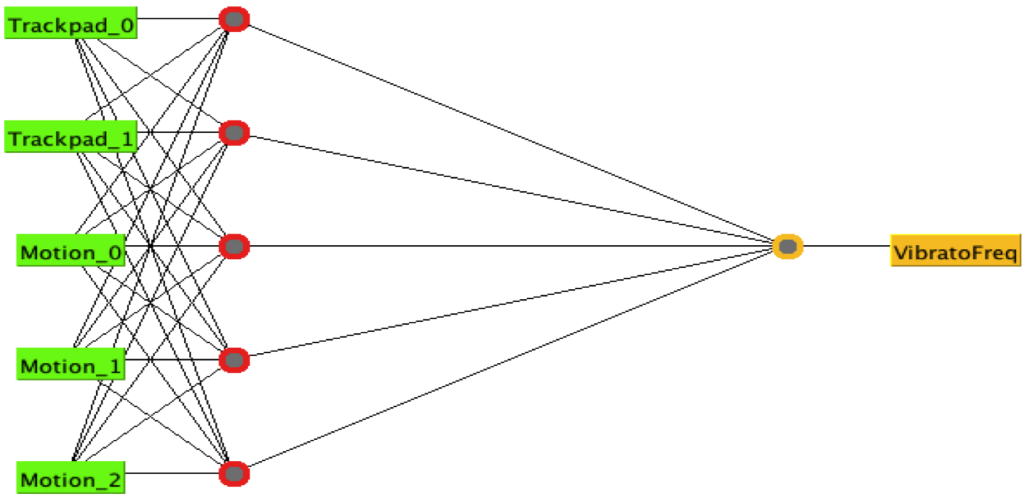


Figure 3.10: The neural network created by default for a continuous-valued parameter ("VibratoFreq") controlled by two trackpad features and three motion sensor features. The network contains five input nodes (green), five hidden nodes in one hidden layer (red), and one output node (orange).

The four discrete classifier algorithms currently supported by the Wekinator are k-nearest neighbor, J48 decision trees, AdaBoost.M1, and support vector machines. These standard algorithms each possess different characteristics regarding their training and classification times, the number and types of parameters they expose to the user, and the types of models they produce.

A k-nearest neighbor (or kNN) classifier produces classification labels using a majority vote of the labels of the k nearest training examples (for some integer $k \geq 1$) in the feature space (Bishop 2007, 125). k is a parameter of the algorithm, and its value affects the degree of smoothing. kNN is the only "instance-based" learning algorithm included in the Wekinator, meaning that its classification computations are performed using the training data directly, rather than requiring a "training" phase to build the model from the data (Russell and Norvig 2003).

J48 is Weka's implementation of the C4.5 decision tree algorithm (Witten and Frank 2005; Quinlan 1993). A decision tree model produces classification labels by a top-down cascade of flowchart-like, "if-then-else" decisions computed on the feature values. The training of a decision tree algorithm produces the model's "tree" architecture and each node's particular decision structure from the training data, then optionally "prunes" away nodes of the tree to simplify the model and reduce overfitting.

AdaBoost.M1 (Schapire 2003) is a multi-class version of the two-class AdaBoost "adaptive boosting" algorithm proposed by Freund and Schapire (1997). AdaBoost is a meta-classifier that uses another classifier as a "weak learner" or "base learner." During training, the base learner algorithm is iteratively trained on variations of the original training dataset to produce a collection of trained models; the AdaBoost algorithm dictates how the training dataset is varied from one iteration to the next,

based on the previously trained models' performance. After training, AdaBoost classifies a new instance using a weighted majority vote of the trained base learner models (these weights are also determined by the AdaBoost algorithm during training). The number of training iterations (and, consequently, the number of trained base learners) is a parameter that may be changed by the user to affect the meta-classifier's classification performance, as well as the time it takes to train. AdaBoost is capable of building a meta-classifier whose generalization accuracy is much better than that of the base learner, and it can use virtually any base learning algorithm, under the condition that the base learner is capable of achieving a minimum accuracy of 50% in each training iteration. In the Wekinator, AdaBoost.M1 boosts on J48 decision trees or decision stumps (one-level decision trees).

A support vector machine, or SVM, is a type of classifier that classifies an instance by computing its distance to a maximum-margin hyperplane in a high-dimensional space (Burges 1998). This space may be of a higher dimension than the feature space, or even of infinite dimension, and the transformation into this space may be nonlinear. SVMs do not represent the decision hyperplane directly, nor explicitly project instances into the high-dimensional space. Instead, distance to the hyperplane is computed implicitly using a "kernel function," computed on the feature values of the instance to be classified and the feature values of a subset of the training instances, called "support vectors." Training an SVM involves defining the maximum-margin hyperplane as a function of support vectors, a convex optimization problem that can be solved using quadratic programming. Many different kernel functions may be used, and the choice of the best kernel is dependent on the learning problem. The Wekinator allows the use of several standard kernel functions: linear, polynomial, and radial basis function (RBF). Weka (and by extension, the Wekinator) uses the sequential minimal optimization (SMO) algorithm for training (Platt 1999) and pairwise classification for extending binary SVMs to multi-class problems (Witten and Frank 2005).

3.3.4 Driving Synthesis Modules and Other Dynamic Processes

The Wekinator can send its models' outputs either to an audio synthesis program written in ChucK or to another process that is capable of receiving OSC messages. If the Wekinator is controlling a ChucK program that implements the Wekinator API for ChucK synthesizers, the user needs only to implement the audio synthesis code, along with methods that will be called by the Wekinator to retrieve information about parameter names and types and to update parameter values. (As discussed in Section 3.3.2 in regard to custom ChucK feature extractors, implementing the custom ChucK synthesizer API involves implementing a class with a specified name and method signatures; Figure 3.11 provides code from an example ChucK synthesis class.) Otherwise, if the process to be controlled is an external OSC process, the user will provide information about its parameter types and ranges via the Wekinator's GUI.

```
public class SynthClass {

	//The simplest sound patch: a single sinusoid
	SinOsc s => dac;

	//This method is called by the Wekinator when its
	//models produce a new parameter vector.
	fun void setParams(float params[]) {
		if (params.size() >= 1) {
			//optionally limit range of parameter
			if (myParams[0] < 0)
				0 => myParams[0];
			if (myParams[0] > 23)
				23 => myParams[0];

			//Update the sounding frequency of the sinusoid
			calcFreqFromParam(myParams[0]) => s.freq;
		}
	}

	//Compute a frequency value from a parameter value
	fun float calcFreqFromParam(float p) {
		//"mtof" converts MIDI note number to frequency (Hz)
		return Std.mtof(72 + p);
	}

	//... Other methods for describing number, types, and names
	// of parameters, turning sound on/off, etc. go here
	//...
}
```

Figure 3.11: Example ChucK code implementing a synthesis class. This synthesis class is a single sine oscillator, and it has one parameter controlling the oscillator frequency.

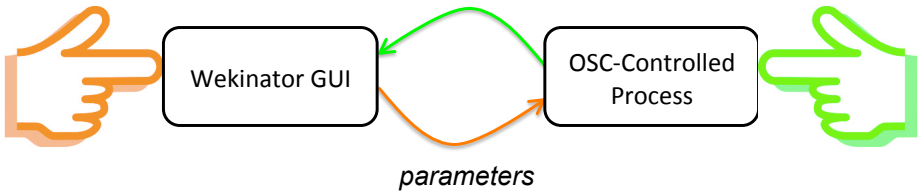


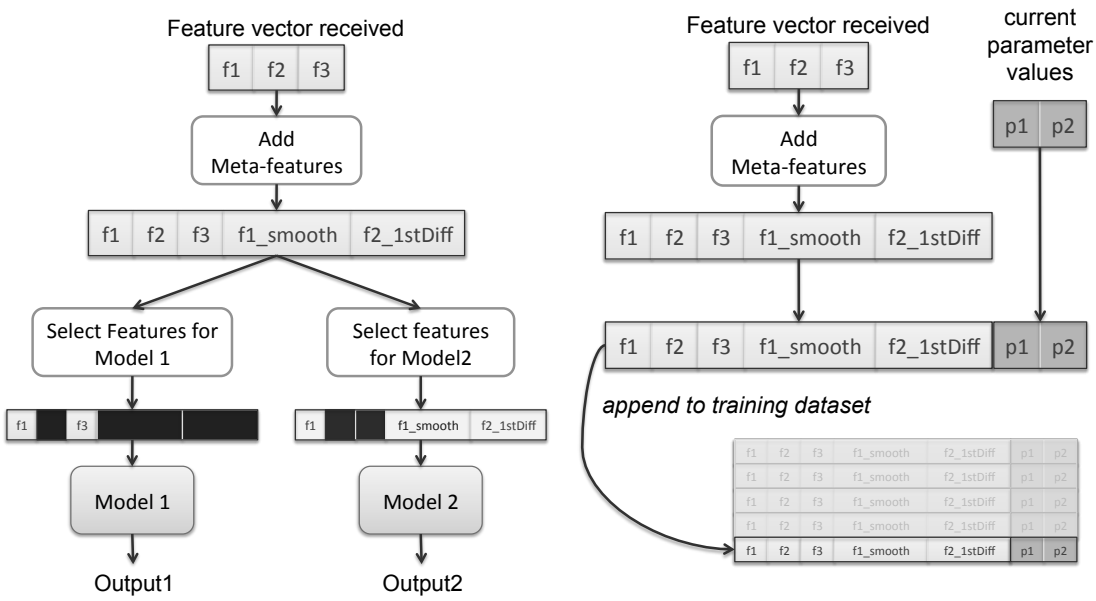
Figure 3.12: OSC communication allows the Wekinator GUI and an OSC-controlled process to maintain a synchronized state. User interactions with the Wekinator GUI trigger a sending of updated parameter values to the OSC process, and changes made to the OSC process directly (e.g., through its own GUI) trigger the sending of a parameter message update to the Wekinator, which updates its state and GUI in response.

The Wekinator can send its models' outputs to any process that is capable of receiving them as an OSC message. This message contains the new parameters as a set of floating-point values, along with the Wekinator-specified message identifier string. Optionally, this process can participate in additional communication with the Wekinator to enable tighter integration with the Wekinator GUI. For example, the Wekinator will send simple control messages via OSC to request the process to turn on or off, in response to user interactions in the GUI. If the OSC process listens to these messages, the user is able to exercise more control over it from the Wekinator. If the OSC process has its own GUI for changing its parameter values, it may communicate its parameter updates back to the Wekinator, so that the current parameter values are always reflected in the Wekinator GUI. This functionality, illustrated in Figure 3.12, can allow the user a more natural means of setting parameter values in the training data creation process: he can experiment with parameter settings from within the process's own user interface, and once he finds a set of parameters he would like to include in a training example, their values will have already been automatically propagated to the Wekinator. He can construct a new training example simply by demonstrating the input feature vector that should correspond to this parameter vector.

3.3.5 The Processing Pipeline: Input Features to Output Parameters

When the Wekinator receives a feature vector, it triggers a cascade of operations that culminates in a model computation (if the user is currently running the Wekinator models on new inputs) or the creation of a new training example (if the user is currently recording training data). This processing pipeline is illustrated in Figure 3.13.

Each time a feature vector is received, it is first processed to compute any user-selected "meta-features." In the current version of the software, the user may instruct the system to compute simple statistics on each individual feature, including the first- and second-order difference and first-order smoothing (implemented as moving 2-point



(a) Receiving a feature when running

(b) Receiving a feature when recording

Figure 3.13: The Wekinator processing pipeline. Receiving a feature vector triggers computation leading to model outputs (3.13a) or a new training example (3.13b). In this example, the learning problem has three features, two meta-features, and two parameters.

rectangular averaging) computed over time, and a "history" buffer of the most recent n values of a feature, for some integer n.

If the user is running the trained models to produce output values from the input features (Figure 3.13a), the feature vector (with any meta-features added) undergoes feature selection for each of the models. By default, every feature is used by every parameter model, but the user may choose to select different subsets of the available features to influence each parameter. Upon receiving its vector of selected features, each model computes its output value.

Alternatively, if the user is currently running the Wekinator to record new training data (Figure 3.13b), the feature vector plus any meta-features is saved to the Wekinator's training dataset, along with the corresponding output parameter values—one per model—that the user has indicated should correspond to that feature vector. Typically, each output parameter will be an integer identifying the discrete class (for classification) or real-valued output (for regression); the output parameter can also be flagged as "not applicable," if the user has indicated that the training example should not be used to train that particular model. This behavior can be used when the user would like to focus on how training data will affect only a subset of parameters. For example, for the learning problem in Figure 3.9, the user might like a specific joystick gesture to result in a specific behavior of the FilterQ parameter, and she would like to record training data to reflect this wish without also thinking about how that joystick gesture will affect the PitchChangeRate parameter. In this case,

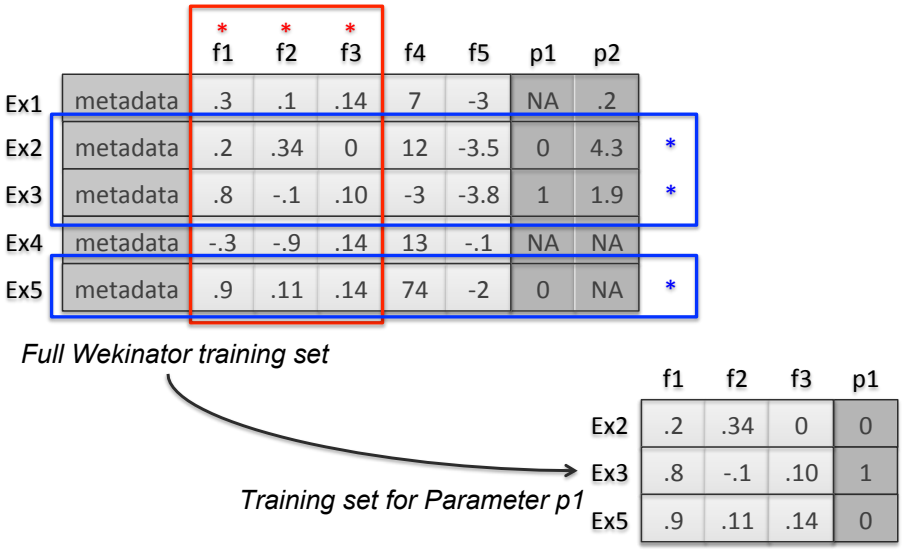


Figure 3.14: A version of the training set is created for each model, using the features and training examples selected for that model. This example shows a training set being created for parameter p_1, using features f_1, f_2, and f_3 (selected for p_1 in the GUI), and all training examples where p_1 is not marked "not applicable" (NA).

she can specify in the GUI that the PitchChangeRate parameter is "not applicable" to the training examples she records while focusing on the FilterQ behavior.

After recording one or more training examples, the user is able to initiate training of the models. At that point, each model is trained on a separate copy of the training dataset, which contains only the features selected for that model, and only the training instances applicable (i.e., *not* flagged as "not applicable") to that model. Figure 3.14 illustrates how the training set for a model is created from the Wekinator's recorded training examples, using a subset of features and examples.

Because each feature vector sent to the Wekinator triggers a computation of the models, a feature extractor might be designed to only output a feature vector to the Wekinator at times when computation is required to supply the controlled process with new parameter values. For example, a system that uses a Wekinator classifier to label the pitch of each note played on an instrument might use a feature extractor with an embedded onset detector that outputs one feature vector every time a note onset is detected. Alternatively, a feature vector might extract features at a constant rate, and the process using the Wekinator's outputs might examine those outputs in order to detect when the pitch of a note has changed. Or, an onset detector might be run on the audio in parallel, and the musical process could take both the onset detector output and the Wekinator output into account in determining its behavior.

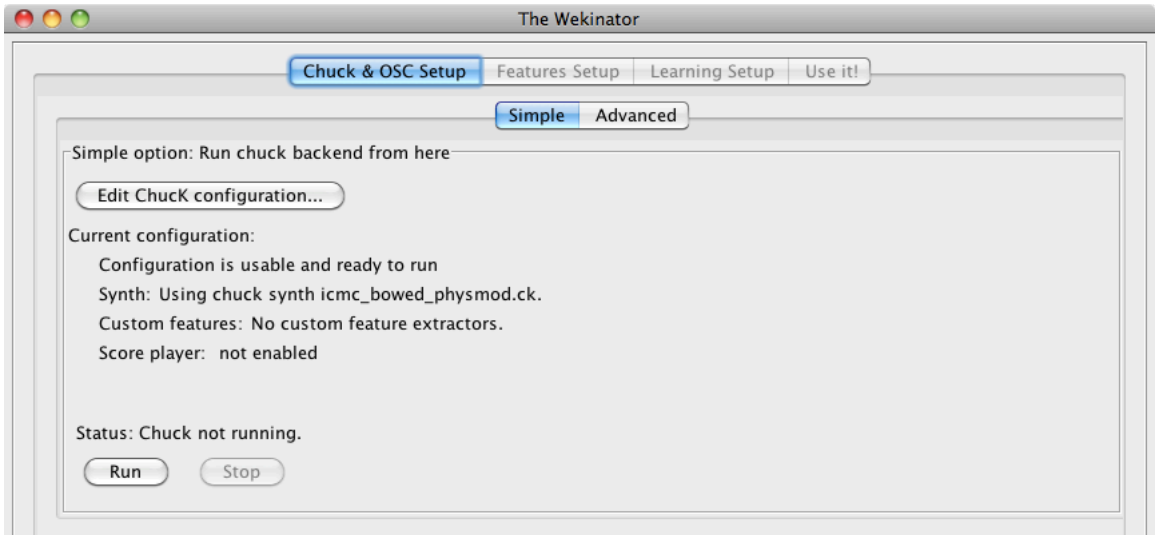


Figure 3.15: The main Wekinator GUI after the application is launched, showing the "Chuck & OSC Setup" tabbed view.

3.4 User Interface and Interaction

In this section, we provide a comprehensive description of the user interface and user interactions in the Wekinator, roughly in the order in which they would be employed by a user.

3.4.1 Application GUI

The main Wekinator user interface is a single window with four tabbed views: "Chuck & OSC Setup," "Features Setup," "Learning Setup," and "Use it!". Figure 3.15 shows these tabs on the Wekinator GUI after the application is first launched. After loading the application, the user is led through the first three of these tabs in sequence, in order to set up the Wekinator for a particular learning problem. In particular, the user must specify which input features will be used, the number and types of parameters to be used and the location to which they will be sent, and the types of learning algorithms to use to build the models. After setup is complete, the user can interact in a free-form manner on the "Use it!" tab to create and edit training data, and to train, run, and evaluate models. The user can also re-visit any of the setup panes to re-configure the Wekinator setup.

3.4.2 ChucK and OSC Setup

When the Wekinator application is launched, only the Java component of the application runs; the ChucK component is configured and launched separately by the Java component, after it is configured to the learning problem. In the first tabbed pane, shown in Figure 3.15, the user configures the ChucK component of the system using

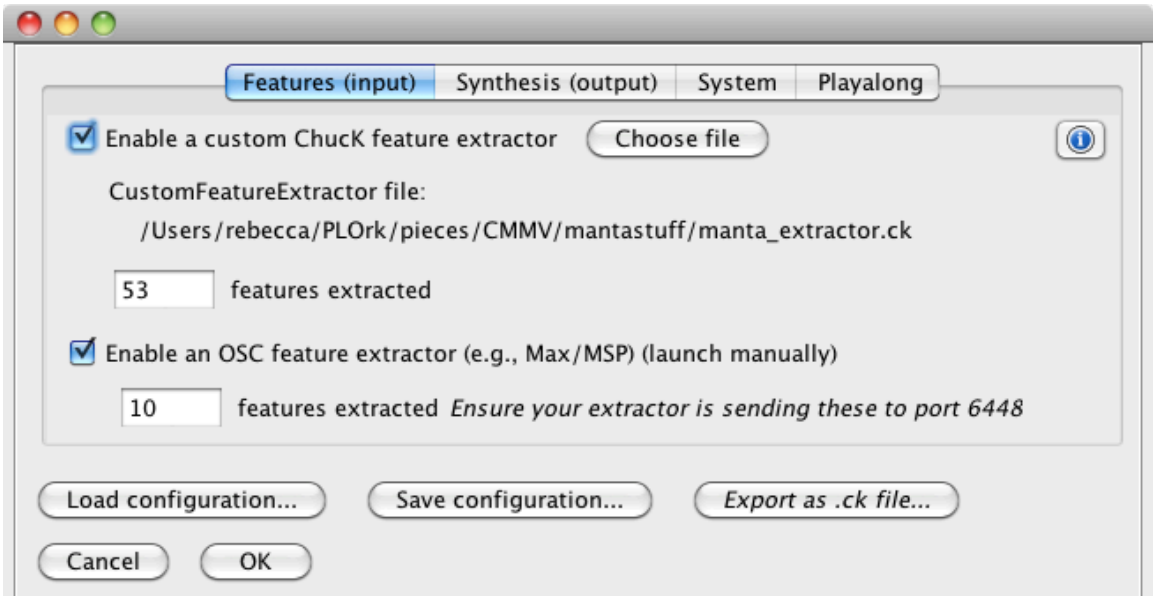


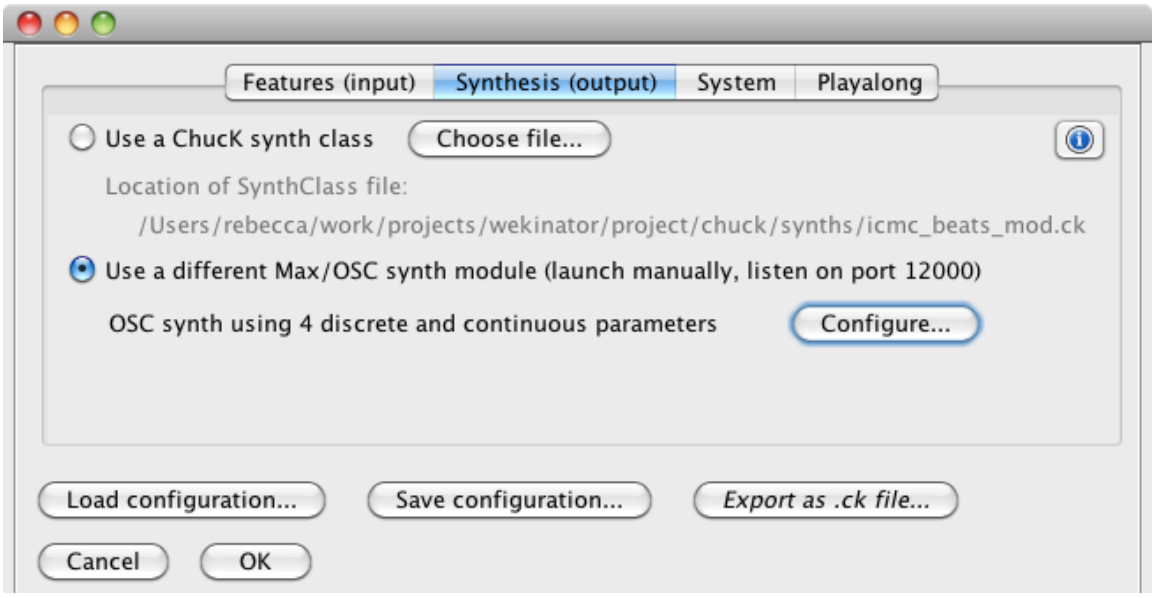
Figure 3.16: The ChucK configuration GUI, shown in a pop-up window when the user clicks on the button "Edit ChucK configuration..." in Figure 3.15. This first tab of the GUI allows the user to configure a custom ChucK feature extractor and/or a custom OSC feature extractor.

the "Edit ChucK configuration" button, then initiates running of the ChucK component and OSC communication between ChucK and Java using the "Run" button.

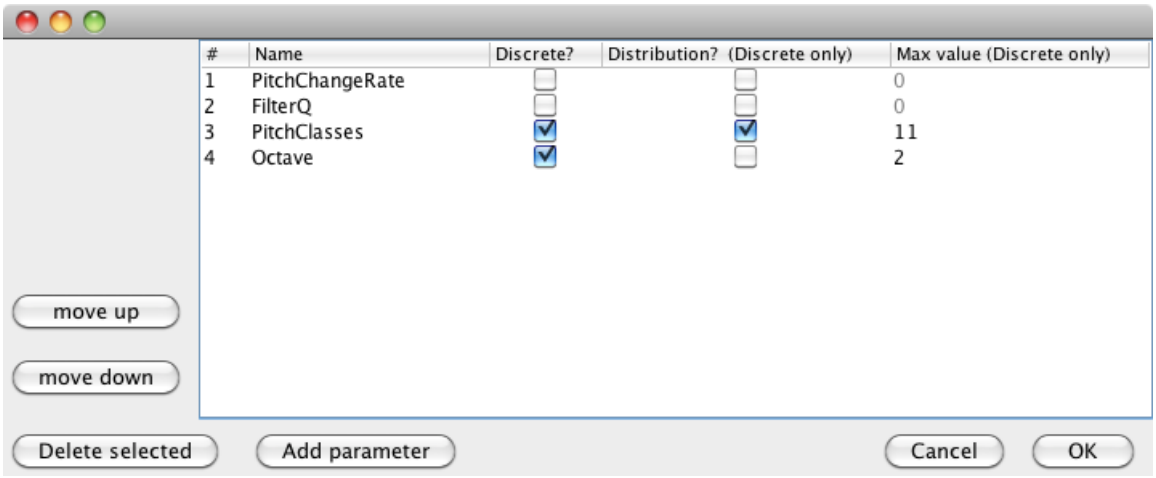
When the user clicks on "Edit ChucK configuration," a pop-up window appears. In the first tab of this window, shown in Figure 3.16, the user specifies whether a user-defined, external ChucK feature extractor will be run (and if so, its file location and the number of features it extracts), and whether a custom, external OSC feature extractor will be run (and if so, the number of features it extracts). In the second tab of this interface, shown in Figure 3.17a, the user specifies whether the Wekinator will send its models' outputs to a ChucK synthesis class (and if so, its file location) or to another process listening via OSC. If the Wekinator is to send its outputs to an OSC process, the user specifies in another pop-up window (shown in Figure 3.17b) the number and names of its parameters, whether each one is continuous or discrete, and for each discrete parameter the total number of classes and whether the Wekinator is to output the best class label or the posterior distribution over all labels.

Both a custom ChucK feature extractor and a custom OSC feature extractor may be run simultaneously; however, the Wekinator will send its outputs to either a ChucK program or an external OSC process, but not to multiple locations. (If the Wekinator is used to control more than one process running independently, for example a ChucK synthesis patch and a Processing animation, the user can implement a process for routing the Wekinator's outputs to all the appropriate locations, and specify that the Wekinator should send its outputs to this routing process.)

Once the user has provided this information, she can start the ChucK component of the system by clicking the "Run" button in the "Chuck & OSC" configuration tab.



(a) Interface for choosing a ChucK or OSC process



(b) Interface for configuring the number, names, and types of parameters for an OSC process. This configuration corresponds to the example problem in Figure 3.9.

Figure 3.17: The second tab of the ChucK configuration GUI, shown in 3.17a, allows the user to choose a ChucK synthesis class or specify that the Wekinator will control an OSC process, which is configured using the interface in 3.17b.

This starts the ChucK virtual machine and runs the necessary ChucK code according to the user-specified configuration. If ChucK is able to successfully run, the GUI brings the user to the next pane, "Features Setup," shown in Figure 3.19. Otherwise, if ChucK encounters an error (for example, if a user-defined ChucK synthesis or feature extractor class cannot be parsed), the user can view information about the error in the Wekinator's Console (Figure 3.18). Also, if ChucK encounters an error later in its running (e.g., an array index out-of-bounds error in the user's code), the user can view information about the error in the Console, and he can use this tab to

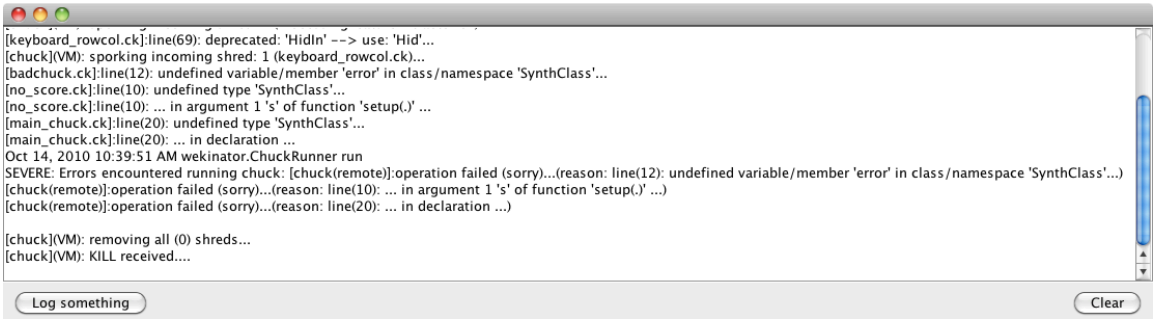


Figure 3.18: The Wekinator error console.

stop and restart the ChucK component of the system independently from the rest of the Wekinator.

When ChucK is run successfully, the current ChucK configuration is saved and will be automatically reloaded the next time the user launches the Wekinator. The ChucK configuration may also be saved to a file and reloaded manually in the future.

3.4.3 Features Setup

The "Features Setup" tab, shown in Figure 3.19, provides an interface for the user to specify which built-in or custom feature extractors will be used. The built-in feature extractors for gestural, audio, and vision inputs have been described in detail in Section 3.3.2 on page 46.

In this interface, the user selects which of the built-in and custom input feature extractors to use via a set of checkboxes; features from multiple sources may be used simultaneously. Several of these features have parameters that may be specified: the user may adjust the rate at which the accelerometer is polled (100 ms by default), as well as the FFT size, window size, window type, and hop size used for audio feature extraction.

To set up a HID device to work with the Wekinator, the user enters a HID configuration interface from within the "Features Setup" tab. Here, he interactively demonstrates the capabilities of the device to the Wekinator by engaging all of the device's axes, hats, and buttons at least once. After demonstration, a message in the GUI indicates to the user the number of axes, hats, and buttons that were detected. The user may optionally save the created configuration and reload it the next time he runs the Wekinator, if he does not want to re-demonstrate the device capabilities.

The user is able to add "meta-features," described in Section 3.3.5, to any of the extracted features via a pop-up window, shown in Figure 3.20. She can select which of the "first-order difference," "second-order difference," "first-order smoothing," or "history" meta-features will be computed for each of the basic features selected in the "Features Setup" tab, and she can specify the length of the "history" meta-feature.

The feature configuration, including which features will be extracted, which meta-features will be extracted, and the HID configuration state, if any, can be saved to a file and reloaded in the future.

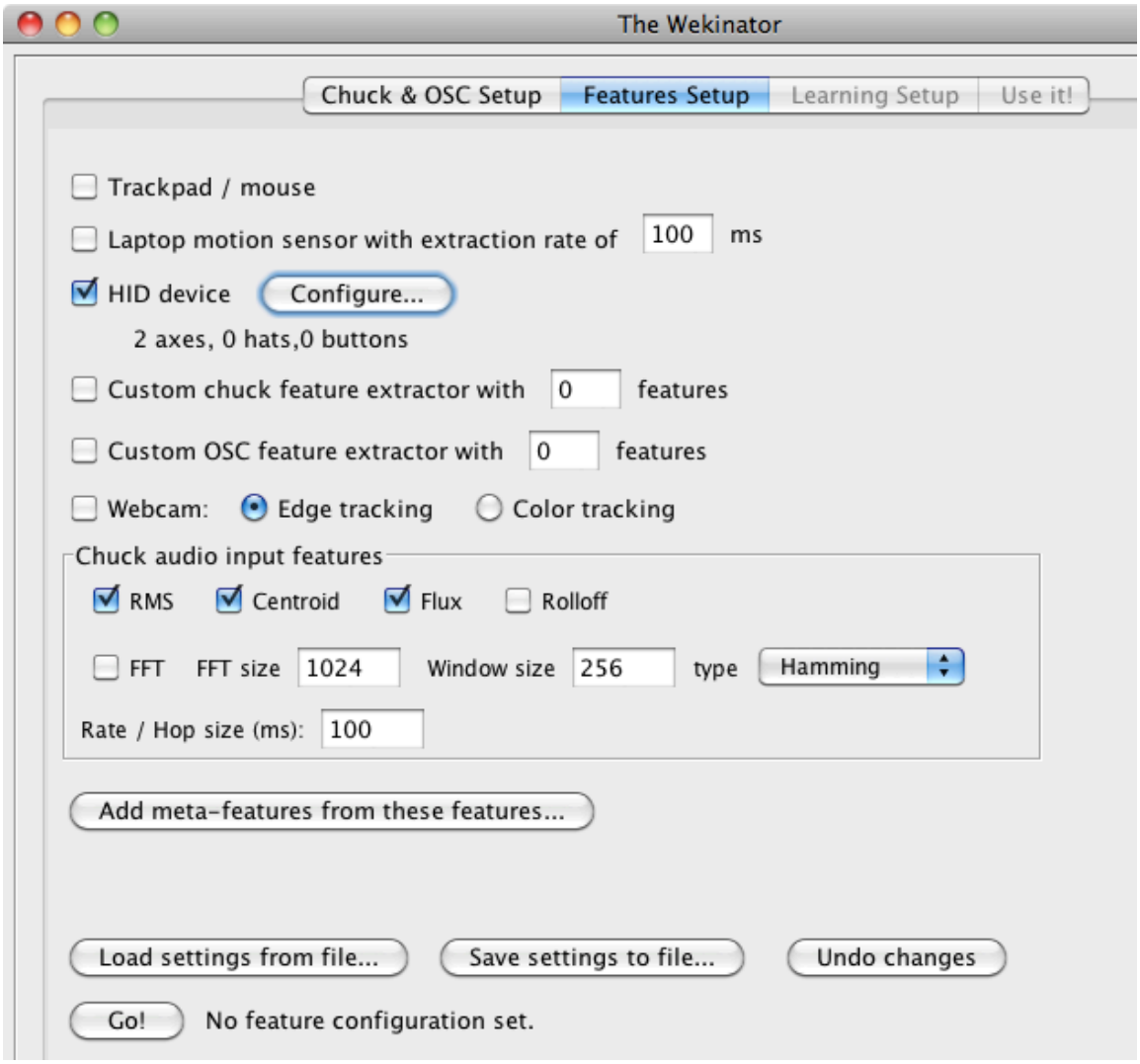


Figure 3.19: The main Wekinator GUI, showing the "Features Setup" tab.

Once the feature setup has been specified, clicking on the "Go!" button in the "Features Setup" tab will take the user to the "Learning Setup" tab.

Also, once the feature setup has been completed, the Wekinator's "Feature Viewer" interface may be used to start and stop feature extraction and allow the user to visually monitor the values of incoming features. The user does not need to use this viewer, but it is available to help with debugging if the user suspects that feature values are not being received or are not being extracted properly. The Feature Viewer can be opened through a top-level Wekinator menu item.

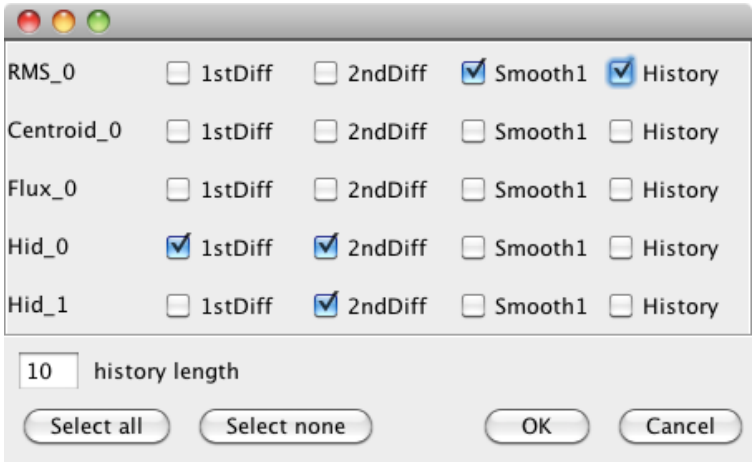


Figure 3.20: The interface for adding one or more "meta-features" to the base features selected in the interface shown in Figure 3.19.

3.4.4 Learning Setup

In the "Learning Setup" tab, shown in Figure 3.21, the user provides information about the learning algorithms that will be used to create the models, the features selected for each model, and the initial training dataset to be used (if any).

The Wekinator automatically creates one independent model for each parameter, as discussed in Section 3.3.3. For continuous parameters, the Wekinator creates a multilayer perceptron neural network. For each discrete parameters, the user is able to pick one of the Wekinator's classification algorithms from a drop-down menu.

By default, all features and meta-features specified in the "Features Setup" tab will be selected for all models; that is, all inputs will potentially affect all models' outputs. The user may change this behavior by clicking on the "View & Choose Features" button for a model, then use a set of checkboxes (shown in Figure 3.22) to indicate which features will be selected for that model. Different models may have different combinations of features selected.

By default, the Wekinator allows the user to record a training dataset for the learning algorithms from scratch, on the "Use it!" tab. However, the "Learning Setup" tab also allows the user to load a pre-existing training dataset from a file. This dataset may have been saved in a Wekinator-specific file format, or it may use Weka's ARFF file format. ARFF files may be created in Weka, Excel, text editors, the Wekinator, or other environments; they are essentially comma-separated value files containing header information describing the features and class value(s) of the dataset.

The training dataset, learning algorithms, and models (trained or untrained) together make up the Wekinator's "learning system," the primary software component that the user interacts with and modifies in the "Use it!" tab and that the user runs when employing trained models in performance. This learning system can be saved to a file and reloaded using this pane.

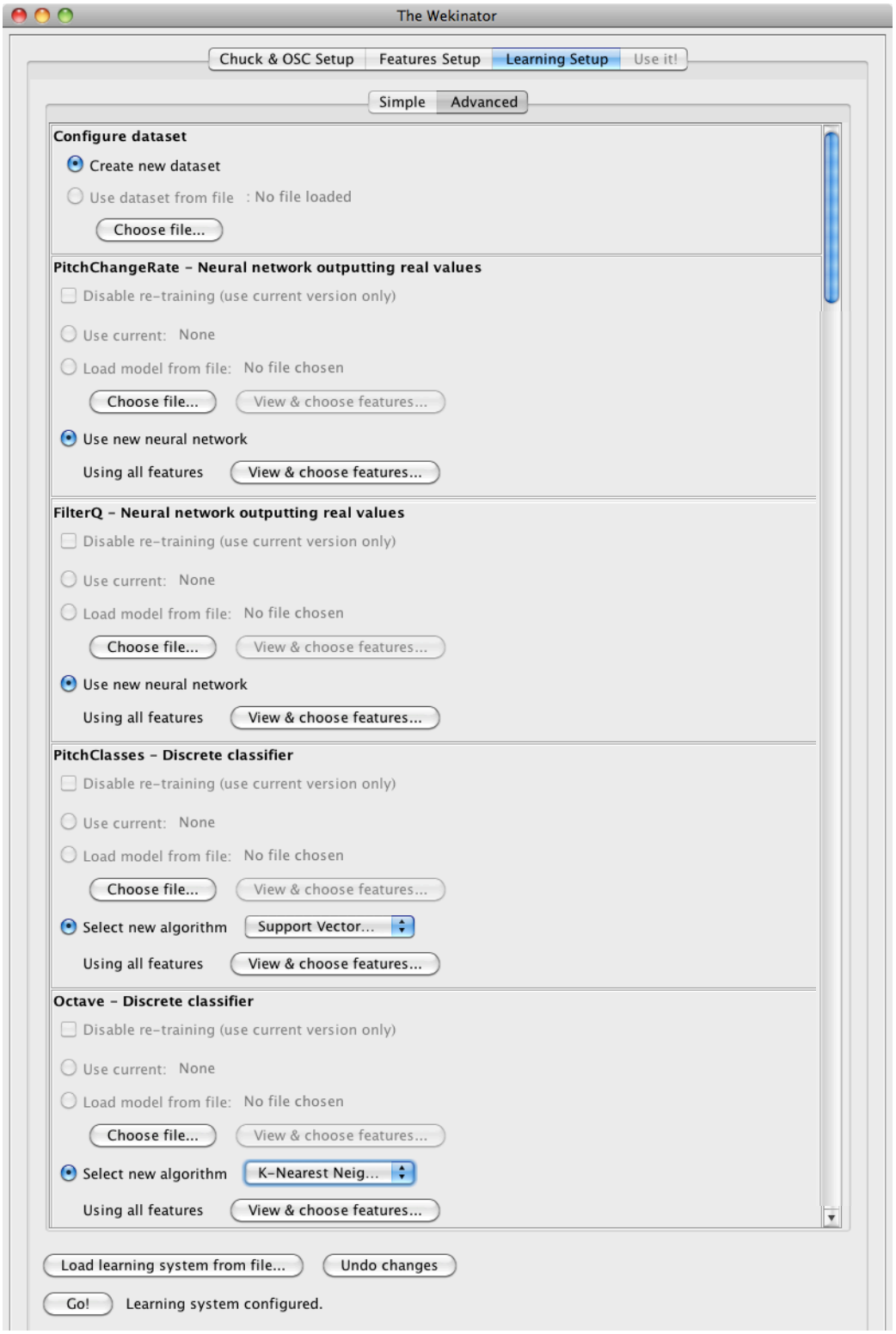


Figure 3.21: The main Wekinator GUI, showing the Learning Setup tab. The setup shown here corresponds to the example learning problem in Figure 3.9.

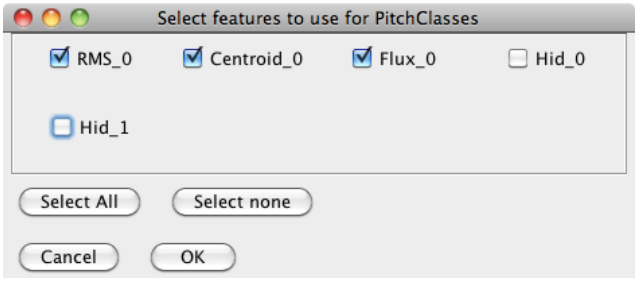


Figure 3.22: The pop-up interface for selecting features for a model. This interface appears in response to clicking a parameter's "View & choose features..." button in the Learning Setup tab (Figure 3.21).

Once the learning setup has been loaded from a file or configured in the GUI, clicking on the "Go!" button of the "Learning Setup" tab will take the user to the "Use it!" tab.

3.4.5 Using the Learning System

The "Use it!" tab, shown in Figure 3.23, has four subviews, each of which is used for a particular type of interaction with the learning system. The user can select the "Collect data," "Train," "Run," and "Configure & Evaluate" subviews in the menu on the lefthand side of the "Use it!" pane. The order in which the data collection, training, running, and configuration and evaluation interactions may occur is constrained only by the learning system state. For example, a model cannot be trained if the training dataset is empty, and the learning system cannot be run if no models are trained. The user will only be able to enter the subviews for the different actions when those actions are legal according to the learning system state. However, users are free to perform any legal action in any order, for example adding data, training, adding more data, retraining, and then running.

Collecting Data

The "Collect Data" subview provides an interface through which the user can interactively create and edit the Wekinator's training dataset.

The "Set parameter values" section at the top of the "Collect Data" subview allows the user to set a value for each of the Wekinator's output parameters. For each real-valued output parameter, the user can enter the value into a text field, and for each discrete-valued parameter, the user can select the value from a drop-down box pre-populated with legal values. The user has the option of sending these values directly to the synthesizer or other process controlled by the Wekinator. This GUI thus provides the user with a means of directly and deterministically controlling the process without the mediation of the parameter models. This is useful, for example, when the user wants to hear or otherwise observe the effect that different parameter settings have on the controlled process.

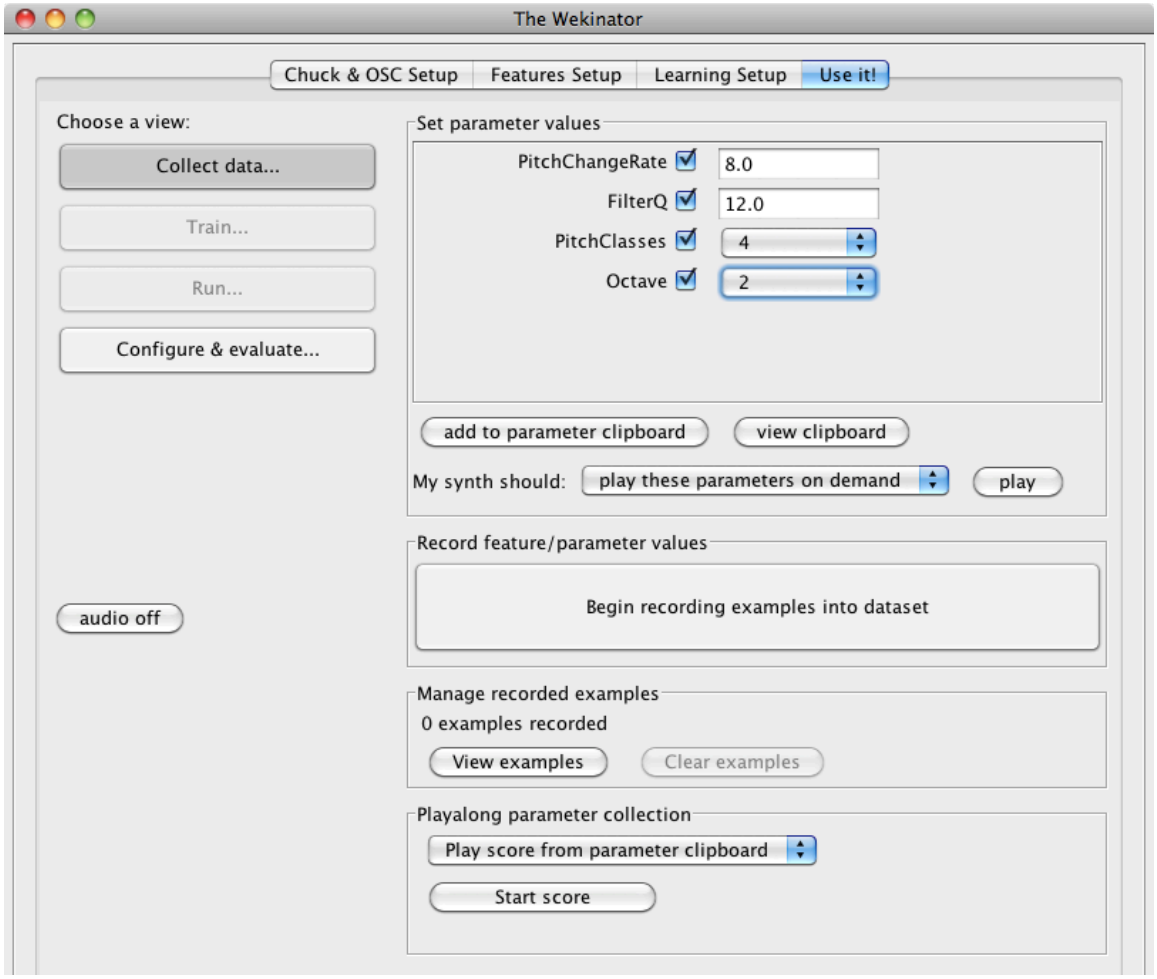


Figure 3.23: The main Wekinator GUI, showing the "Use it!" tab's "Collect data" subview. This interface is used to record new training data.

When the user clicks the "Begin recording examples into dataset" button, the Wekinator causes all built-in feature extractors to begin extracting features, and the Wekinator enters a RECORDING state, in which all incoming feature vectors are processed and added to the training dataset as discussed in Section 3.3.5 and illustrated in Figure 3.13b. Each time a feature vector is received and processed, it will be added to the training set along with the parameter values currently specified in the "Set parameter values" portion of the Wekinator GUI. Therefore, in order to create training data, a user can set the parameter values in the GUI, click on the recording button, then demonstrate the gestural, audio, or other inputs that she would like to correspond to those parameters. Alternatively, if the Wekinator is being used to drive an OSC process with its own user interface for setting parameter values, that process can use OSC to communicate any changes in its parameters back to the Wekinator, as mentioned in Section 3.3.4 on page 54. In that case, the "Set parameter values" portion of the Wekinator GUI will be automatically updated to reflect the current parameter values.

By default, each new instance is saved to the training dataset with all the parameter values specified in the "Set parameter values" portion of the "Collect data" subview. However, in the case that the user does not want the training instances being recorded to affect one or more specific parameters (that is, she does not want them to be used in the training of those parameters' models, as discussed in Section 3.3.5), she can specify that the data being recorded is not applicable to a parameter by unchecking the checkbox appearing next to the parameter value name. This will cause each new instance to be stored in the Wekinator training set with a "not applicable" value for that parameter.

The Wekinator will stop recording when the record button (whose text label is changed to "Stop recording" when the Wekinator is in the RECORDING state) is clicked again. This also causes the running built-in feature extractors to stop extracting features.

Because each incoming feature vector results in a new training instance as long as the Wekinator is in the RECORDING state, more than one feature vector is typically recorded at a time for each gesture that the user demonstrates. Built-in, non-audio feature extractors are extracted at a rate of 20Hz, and audio feature extractors use a default rate of 10Hz, so a sizable training set can be built up quite quickly. This behavior allows a user to easily create training instances whose parameter vector values are the same but whose feature vectors are different. For example, to create a gesture classifier whose output should be robust to small changes in the input gesture, a user can subtly change his gesture over time while recording training instances for the same set of parameter values. Also, this behavior allows the user to give certain gestures more "weight" or importance in creating the models: by recording particular gestures for longer amounts of time, and therefore creating more instances of that gesture in the training dataset, the learning algorithm can be induced to implicitly treat this gesture as more important when creating the model.

In addition to the "Begin recording examples into dataset" GUI button, the Wekinator offers two alternative mechanisms for starting and stopping the RECORDING state. The first is by using a foot pedal, pictured in Figure 3.24. This pedal allows the user to control the recording process while performing gestures that require both hands, for example. The other mechanism is by OSC control messages that allow any other process to start and stop the Wekinator's RECORDING state. This allows recording to be controlled from a different GUI or an alternative interface, for example a musical collaborator sending control signals over a wireless network.

During recording, as new feature vectors are received and new instances are added to the Wekinator's training dataset, the "Manage recorded examples" portion of the "Collect Data" subview shows the user how many examples have been recorded, in total. By clicking on the "View examples" button, the user can view these examples in a spreadsheet-like interface (Figure 3.25), in which each row is a unique training example and each column is either a feature, a parameter, or a meta-data field. The Wekinator stores meta-data describing each example's ID (a unique identifier), the time at which it was recorded, and the "recording round" in which it was recorded. Each time the Wekinator enters the RECORDING state is considered a new recording

Figure 3.24: The Footime USB controller, produced by Bili Inc., used for hands-free control of Wekinator recording and running.

ID	Time	Recording round	RMS_0	Centroid_0	Flux_0	Hid_0	Hid_1	PitchChangeRate	FilterQ	PitchClasses	Octave
0	14:26:13:591	1	0	0.078	0	-0.004	-0.004	2.0	2.9	0.0	0.0
1	14:26:13:649	1	0	0.069	1	-0.004	-0.004	2.0	2.9	0.0	0.0
2	14:26:13:696	1	0	0.069	1	-0.004	-0.004	2.0	2.9	0.0	0.0
3	14:26:13:748	1	0	0.081	0.533	-0.004	-0.004	2.0	2.9	0.0	0.0
4	14:26:13:800	1	0	0.081	0.533	-0.004	-0.004	2.0	2.9	0.0	0.0
5	14:26:13:846	1	0	0.094	0.338	-0.004	-0.004	2.0	2.9	0.0	0.0
6	14:26:13:899	1	0	0.094	0.338	-0.004	-0.004	2.0	2.9	0.0	0.0
7	14:26:26:427	2	0	0.051	0.814	-1	-1	3.4	32.0	1.0	0.0
8	14:26:26:479	2	0	0.039	0.051	-1	-1	3.4	32.0	1.0	0.0
9	14:26:26:525	2	0	0.039	0.051	-1	-1	3.4	32.0	1.0	0.0
10	14:26:26:577	2	0	0.053	0.032	-1	-1	3.4	32.0	1.0	0.0
11	14:26:26:630	2	0	0.053	0.032	-1	-1	3.4	32.0	1.0	0.0
12	14:26:26:676	2	0	0.049	0.013	-0.004	-1	3.4	32.0	1.0	0.0
13	14:26:26:728	2	0	0.049	0.013	-0.004	-1	3.4	32.0	1.0	0.0
14	14:26:26:775	2	0	0.038	0.024	-0.004	-1	3.4	32.0	1.0	0.0
15	14:26:26:827	2	0	0.038	0.024	-1	-1	3.4	32.0	1.0	0.0
16	14:26:26:879	2	0	0.035	0.078	-1	-1	3.4	32.0	1.0	0.0
17	14:26:36:440	3	0	0.06	0.828	1	-1	3.4	32.0	?	?
18	14:26:36:493	3	0	0.057	0.077	1	-0.004	3.4	32.0	?	?
19	14:26:36:539	3	0	0.057	0.077	1	-0.004	3.4	32.0	?	?
20	14:26:36:591	3	0	0.066	0.365	1	-0.004	3.4	32.0	?	?
21	14:26:36:643	3	0	0.066	0.365	1	-0.004	3.4	32.0	?	?
22	14:26:36:690	3	0	0.055	0.217	1	-1	3.4	32.0	?	?

Delete selected | Add row | Listen | Save to ARFF... | Done

Figure 3.25: The Wekinator's spreadsheet-style interface for viewing and editing training data.

round, so the "recording round" meta-data field can be useful when a user would like to edit or delete all examples recorded in the same round.

In the spreadsheet interface, the user may edit the meta-data, features, or parameters of any training example. The user can also manually add new training examples and delete examples. From this interface, and from a Wekinator menu item, the user may also save the training dataset to an ARFF file for future use in the Wekinator, Weka, or elsewhere. When certain training examples have been specified as "not applicable" to the training of a particular parameter model (using the checkboxes in the "Set parameter values" section of the "Collect Data" subview), those parameter values for those instances appear in the spreadsheet as "?" values. For example, training examples with ID values from 17 to 22 in Figure 3.25 were recorded as "not applicable" to parameters "PitchClasses" and "Octave." The user can manually enter

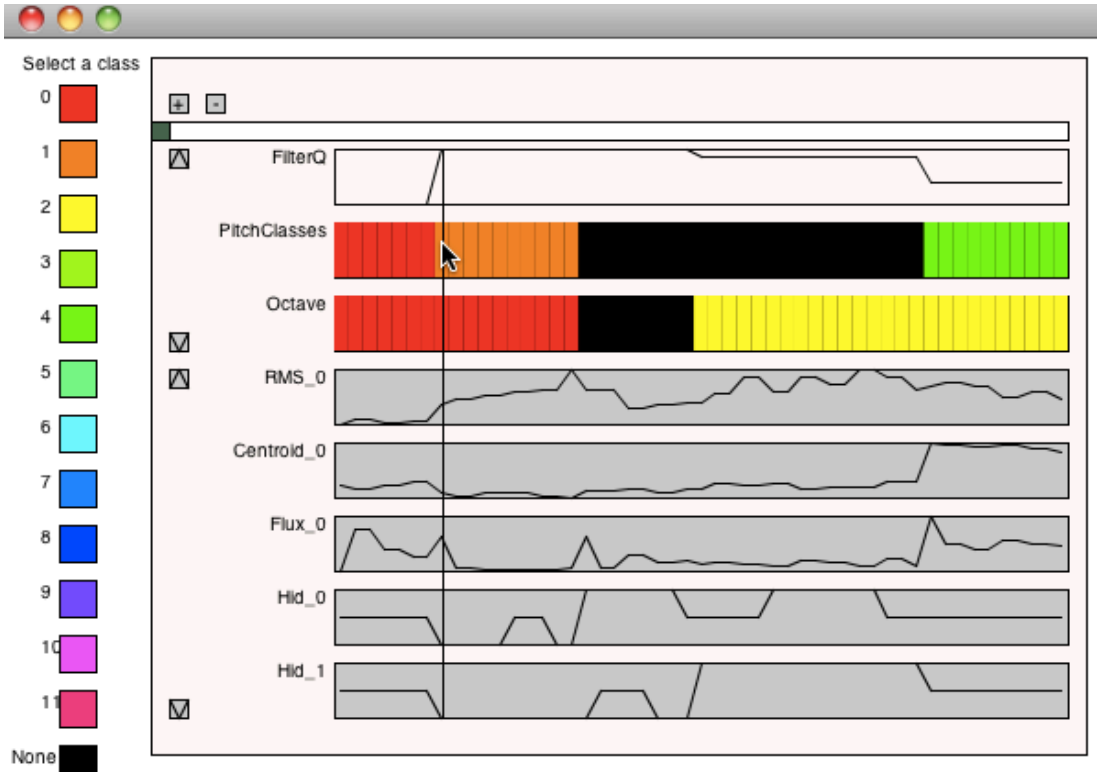


Figure 3.26: The Wekinator's graphical interface for viewing and editing training data. Different colors of each discrete parameter track at the top indicate different classes. The user can zoom, scroll, and edit class values using the mouse and the colored class palette on the left.

"?" values and change "?"s to numeric values in the spreadsheet viewer to change this behavior.

The Wekinator provides an alternative, graphical interface for visualizing and editing the training dataset. This interface is shown in Figure 3.26. It presents information about feature and parameter values using a track-based interface, similar to that used in audio recording, where each feature or parameter appears in a separate track. In these tracks, the value of each feature and each parameter is plotted over time, in the order in which they were recorded into the dataset. The user can zoom in and out to view larger or smaller segments in time, as well as scroll horizontally to view earlier or later in time. Continuous-valued parameters and all features are plotted as "waveform"-like lines, and discrete-valued parameters are displayed as units of color, one unit per instance, where the color indicates the class of the instance. Users can change the class of instances by selecting a range of consecutive instances with the mouse and clicking on a new class value in the "palette" on the left. Instances specified as "not applicable" to a particular model will be colored black in that model's track. The user can changed colored units to black and vice versa.

The user may quickly delete the entire training dataset using the "Clear examples" button in the "Collect data" subview.

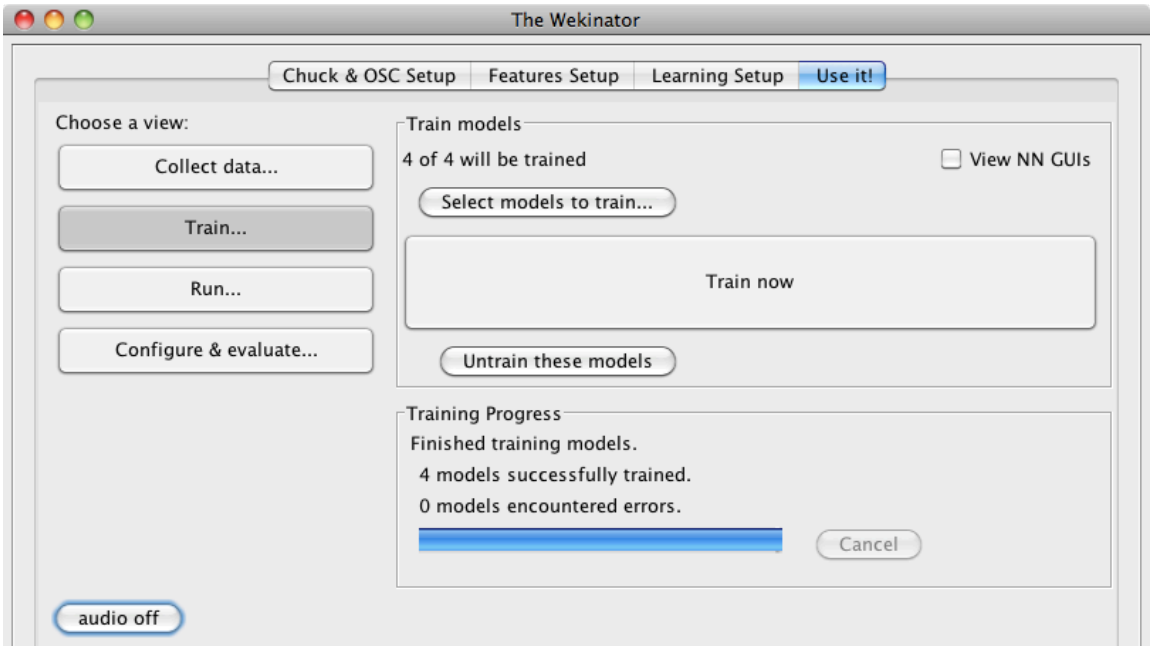


Figure 3.27: The "Train" subview of the Wekinator's "Use it!" tab. In this interface, the user selects models for training, initiates training, and is informed of the training progress.

This concludes the discussion of the first set of interfaces for creating and editing training data; the second mode of creating training data, called "playalong example recording," is discussed later in Section 3.4.6.

Training

Once the training set contains data, the user may train a model for one or more parameters using the "Train" subview of the "Use it!" tab, shown in Figure 3.27. When the user clicks the "Train now" button, a separate training dataset is constructed for each model, including only the features selected for that model, and including only the training examples applicable to that model (Figure 3.14). Then, each model is built from the training data using the algorithm chosen for that parameter in the "Learning Setup" tab.

It is also possible to initiate training by sending an OSC control message to the Wekinator from some other program.

As each model trains, a progress bar informs the user of the progress of the training operation. The user is able to cancel the training at any point. A status message informs the user of the number of models trained and the number of models that encountered errors (if the algorithm was unable to build a model from the data for some reason).

By default, all models are trained from the entire set of applicable training data every time the "Train now" button is clicked. If a user does not want certain models to be trained, for example because the training data applicable to those models has

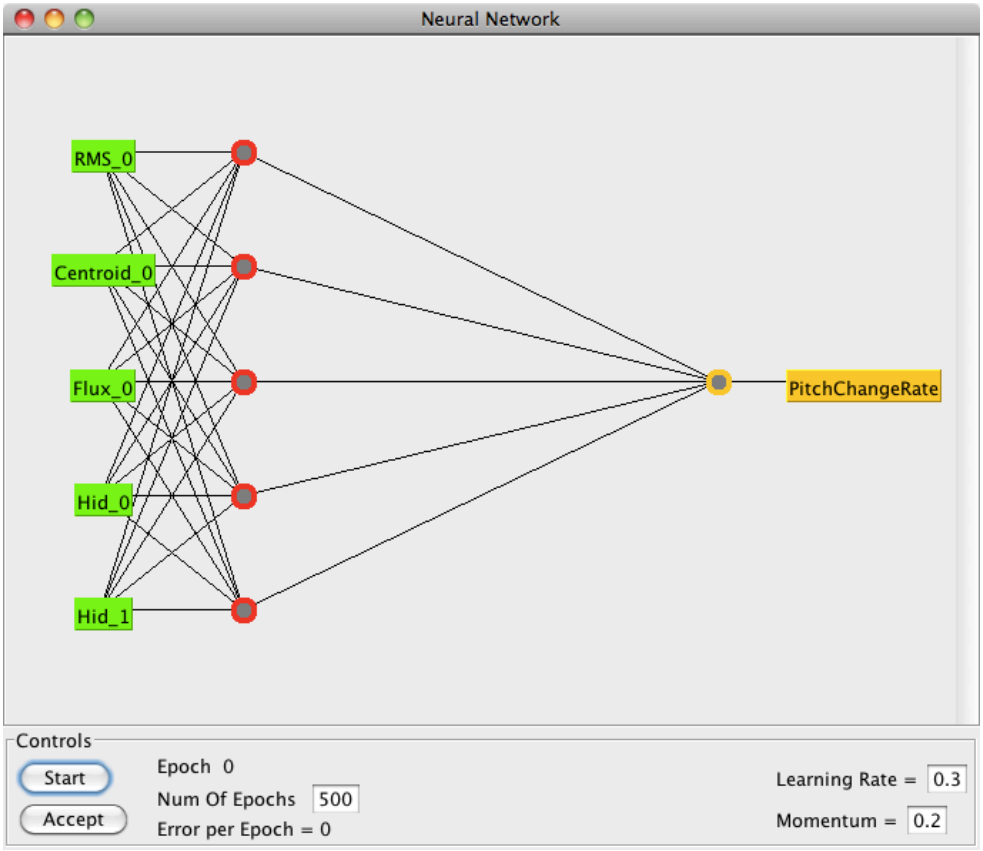


Figure 3.28: The GUI for editing the neural network architecture, changing training parameters, and controlling the training process. This GUI is implemented in Weka and optionally displayed to the Wekinator user for each multilayer perceptron model.

not changed, he can indicate in a pop-up box (opened by the "Select models to train" button) which of the models should be trained at that time.

If one or more of the parameters is continuous, the user will have the option of viewing a GUI for controlling the multilayer perceptron training process. This option is selected using the "View NN GUIs" checkbox in Figure 3.27. This neural network control GUI, shown in Figure 3.28, is implemented in Weka itself. The GUI allows the user to edit the architecture of the network, start and cancel training of the network, and change several parameters controlling the training procedure (the number of training epochs, learning rate, and momentum).

Running

Once the models have been trained, the user may run them in real-time on new inputs using the "Run" subview of the "Use it!" tab, shown in Figure 3.29. When the "Run" button is clicked, the Wekinator enters into a RUNNING state and all built-in feature extractors begin extracting features. As shown in Figure 3.13a on page 57, each time a new feature vector is received, it is processed and sent to each

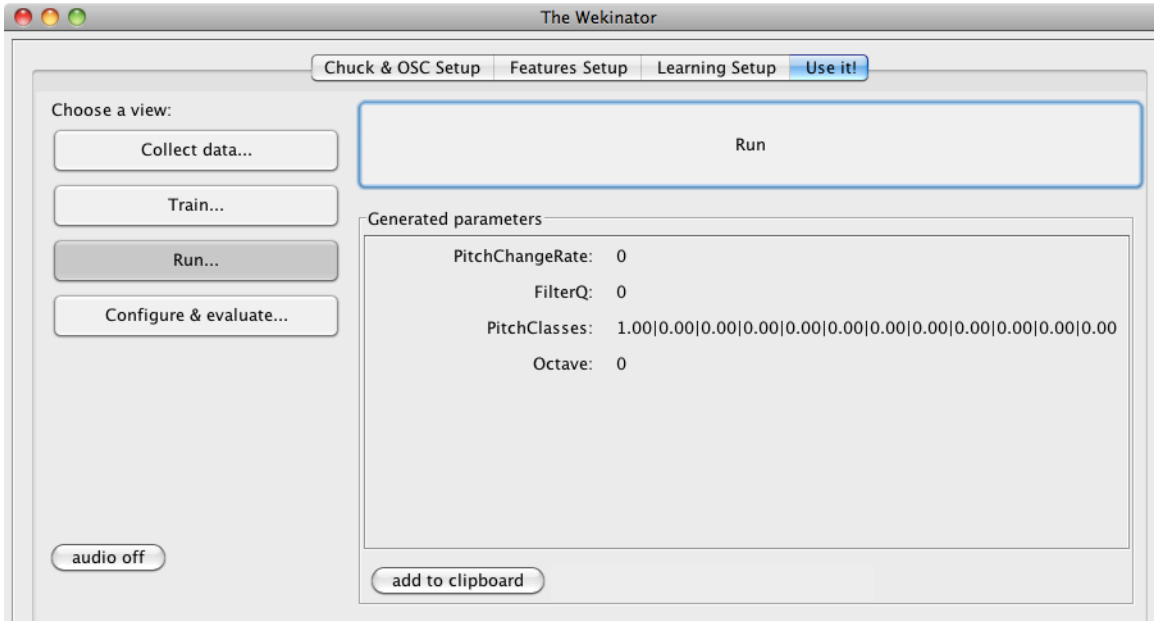


Figure 3.29: The "Run" subview of the Wekinator's "Use it!" tab. In this interface, the user starts and stops running of the trained models on new, real-time inputs. Models' output values (or posteriors) are displayed textually, as well as being sent to the ChucK or OSC process as control parameters.

of the trained models. Once each of the models has produced an output value (or posterior distribution) from the feature vector, the vector of outputs is sent to the ChucK synthesizer or the OSC process. The model values are also displayed textually in the GUI. When the "Run" button (whose text label changes to "Stop" when the Wekinator is RUNNING) is clicked again, the Wekinator exits the RUNNING state and the built-in feature extractors stop extracting features.

It is also possible to control entering and exiting the RUNNING state using the foot pedal in Figure 3.24 or by sending OSC control messages to the Wekinator.

As the Wekinator runs, the model output vector drives the synthesis algorithm or other dynamic process in real-time, in response to features being extracted in real-time. This allows the user to evaluate the set of trained models in a hands-on way, for example providing new gesture or sound inputs in real-time and observing how the synthesis algorithm or other process being driven by the Wekinator responds. The model outputs are also displayed textually in real-time, in the "Generated parameters" section of the "Run" subview.

Configuring and Evaluating

The final subview of the "Use it!" tab is the "Configure & Evaluate" view, shown in Figure 3.30. This view allows users to change learning algorithms' parameters and evaluate one or all models' training and cross-validation accuracies (see Section 2.2.1 for a discussion of these metrics).

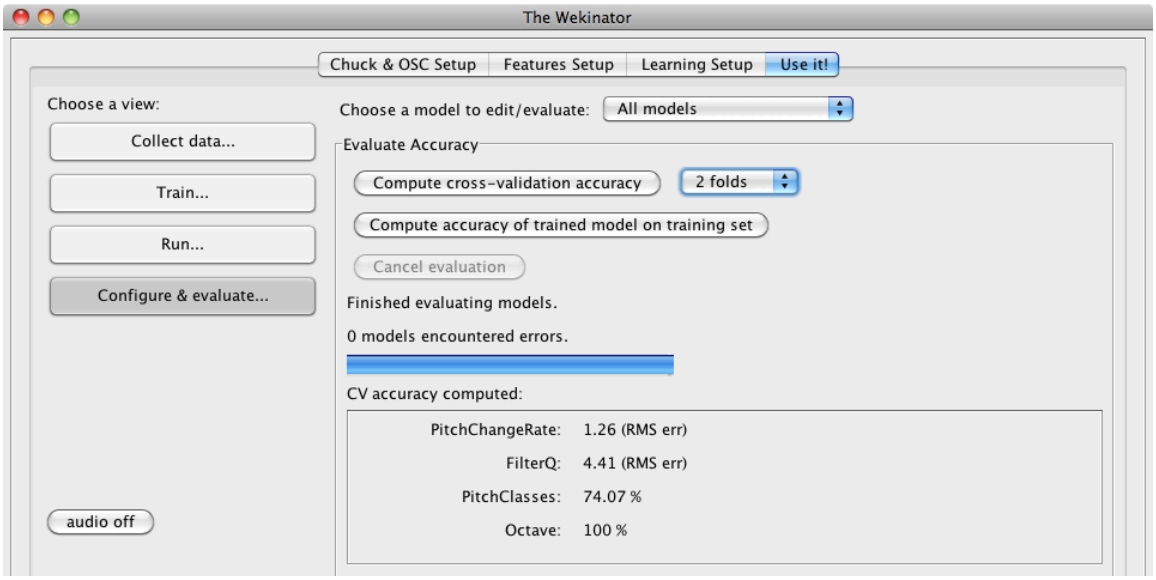


Figure 3.30: The "Configure & Evaluate" subview of the Wekinator's "Use it!" tab. In this interface, the user is able to configure individual models and evaluate models individually or as a set, using cross-validation and training accuracy. This figure shows the view for evaluating all models as a set.

A drop-down box in this view enables the user to evaluate the set of models, or to drill-down on individual models. To evaluate all models, the user can choose to compute their training accuracy or 2-, 5-, or 10-fold cross-validation accuracy. In each case, computation is performed on each model's version of the training set, with only selected features and instances included. A progress bar indicates the progress toward completion as each model is evaluated, and the GUI displays the computed accuracy (in the case of discrete classifiers) or RMS error (in the case of multilayer perceptron neural networks). The user is able to cancel evaluation if it takes too long.

By selecting to drill down on a particular model, a user can evaluate the cross-validation accuracy or training accuracy (or RMS error) for just that model. A user can also change the parameters of the learning algorithm used to build that model. The parameters available for each model are summarized in Table 3.1. Figure 3.31 shows a drill-down interface for an SVM model.

The user is also able to save the trained model as a Wekinator file or as a serialized Weka classifier, which can be reloaded into Weka or used in Java code built with the Weka libraries. The user is also able to change the learning algorithm and features used to build a model by revisiting the "Learning Setup" tab.

3.4.6 Playalong Example Recording

In Section 3.4.5 above, we discussed how training examples may be created by specifying a single set of parameter values in the Wekinator GUI then recording feature values as the user demonstrates inputs (e.g., a gesture) intended to correspond to

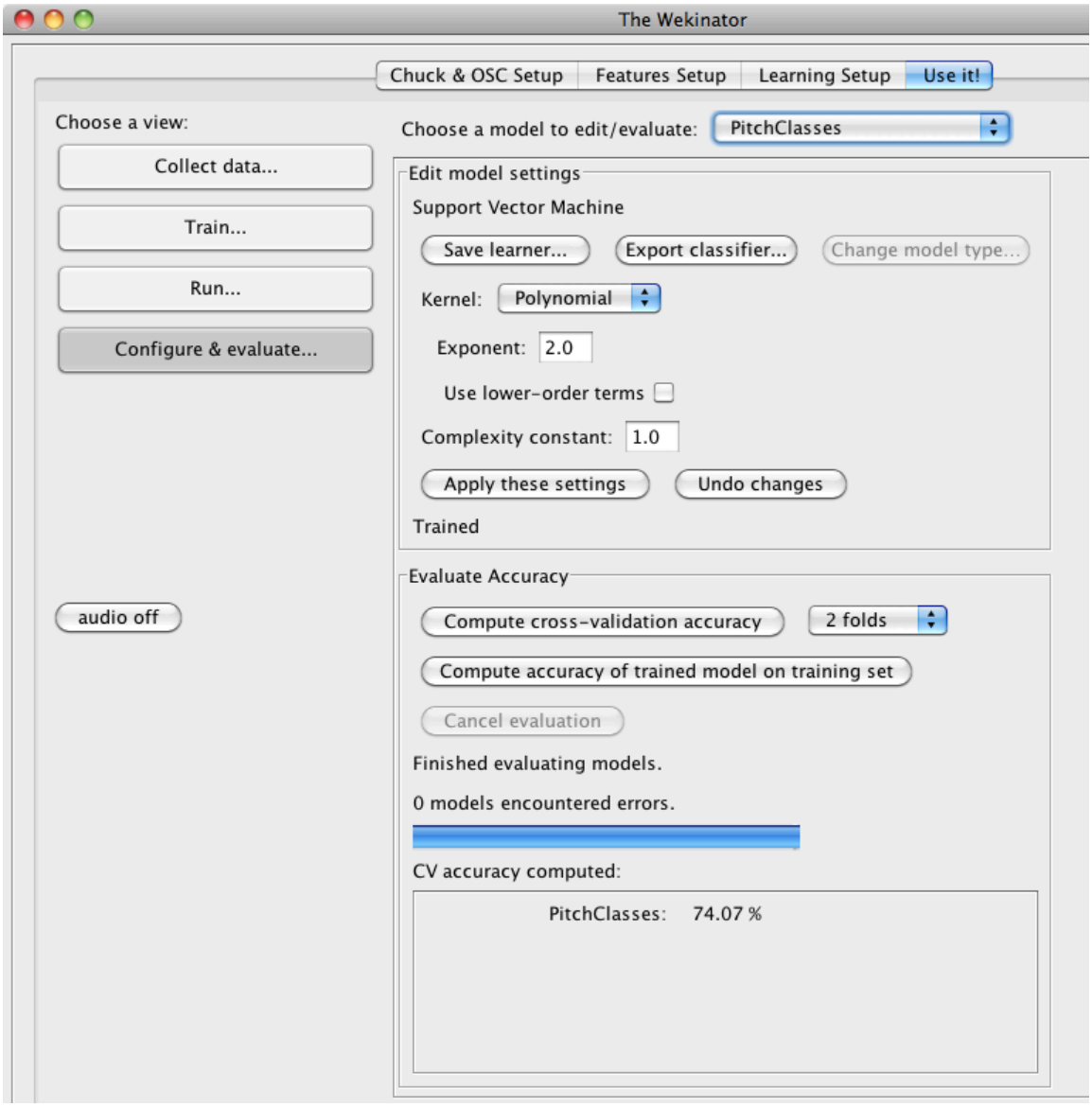


Figure 3.31: The single-model drill-down interface of the "Configure & Evaluate" subview, shown here for the "PitchClasses" parameter using an SVM algorithm. This interface allows a user to modify and evaluate the algorithm for a single model.

those parameter values. The Wekinator offers another mode of creating training examples, which we call "playalong" example recording.

The motivation for playalong example recording is to enable a more natural method of training data creation, in particular allowing the user to consider how changes in the input signals over time correspond to changes in the parameters over time. In a musical instrument creation application with continuous sound synthesis parameters, for example, an instrument builder might not only be concerned with obtaining particular sounds in response to particular physical gesture states, but also

with matching evolving performer gestures with evolving sonic gestures in a musically expressive way.

In playalong recording, the parameters of the Wekinator-controlled process are driven according to a "score." Like a musical score, the Wekinator score dictates how the parameter values change over time. In playalong mode, the parameters are set only according to the score being played; the user-provided input features have no influence over the parameters. As the user provides input features that change over time, in sync with the the score-driven parameters, "snapshots" of the current feature and parameter vectors are added to the training dataset as new examples. Using playalong functionality, our example instrument builder might set up a score that plays through a sequence of sounds that he'd like his new instrument to be able to replicate. He is able to create training data by by gesturing along with the score *as if he were controlling the sound* in real-time, in a manner akin to how someone might play "air guitar" or lip-sync along with a recorded song.

Playalong scores may be encoded in a pre-written ChucK program or generated on-the-fly by the user, within the Wekinator GUI. A ChucK score file may be designated in the Chuck configuration step (discussed in Section 3.4.2), using the "Playalong" configuration tab of the ChucK configuration window. This score file must implement a ChucK class named "ScorePlayer," containing a method "playScore" that updates the parameters over time. The playScore method may set these parameters in a deterministic manner, for example to play a melody, or it may change their values and/or timing using a stochastic process, for example executing a random walk through the parameter space.

In either case, ChucK offers a particularly concise syntax for specifying these parameter changes over time: in ChucK, time and duration are primitive types, and the ChucK "=>" operator allows sub-sample-level control over the timing of events. For example, the ChucK playScore method in Figure 3.32 increments a parameter value every 5 audio samples (approximately every 0.11 ms). Furthermore, because of the timing guarantees imposed by the ChucK virtual machine, the parameter changes driven by a ChucK score are guaranteed to remain sample-synchronous with all running ChucK synthesis and feature extraction code.

ChucK scores may optionally implement a series of text strings to be displayed to the user within the Wekinator GUI, which are updated over time in sync with the parameter changes. For example, the message may inform the user which parameter values will be next, and when they will be set, so that he is better prepared to react to them.

The user has the ability to construct deterministic scores on-the-fly from the Wekinator GUI. The Wekinator has a "parameter clipboard" consisting of one or more parameter vectors, each associated with a duration over which those parameter values should hold. By clicking the "Add to parameter clipboard" button in the "Collect data" subview of the "Use it!" tab (Figure 3.23 on page 67), the current values displayed in the "Set parameter values" portion of the GUI are appended to the clipboard with a default duration of 1 second. This allows the user to dynamically populate the clipboard with parameter values of his choice. Additionally, the user can

```
while (isPlaying) {
    i++ => p[0]; //increment 1st parameter
    mySynth.setParams(p); //set parameters
    5::samp => now; //wait 5 samples
}
```

Figure 3.32: ChucK code that, when placed inside a playScore method of a ScorePlayer class, increments the value of the first parameter every 5 samples.

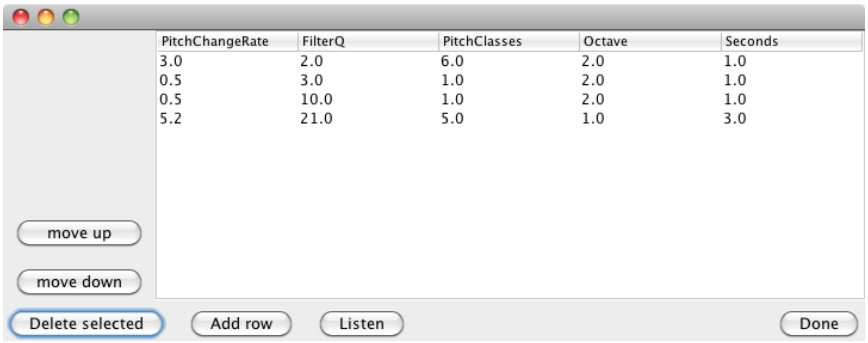


Figure 3.33: The playalong clipboard. Each row corresponds to a set of parameters and an associated duration. Playing the clipboard will set the parameters in sequence, from top to bottom, maintaining each set of values for the specified duration of time.

capture the current parameters being played in the RUNNING state, using the "add to clipboard" button in the "Run" subview (Figure 3.29 on page 73).

The parameter clipboard may be viewed using the "view clipboard" button on the "Collect data" subview. An example parameter clipboard containing four rows of parameters is shown in Figure 3.33. Using this viewer, the user can add and delete parameter rows, re-order parameter rows, edit parameter values and durations, and send the parameter values of a row out to the ChucK or OSC process being controlled (using the "Listen" button).

The "Playalong parameter collection" portion of the "Collect data" interface (Figure 3.23) allows a user to start and stop a playalong score, and to choose whether to play a ChucK score or the score in the parameter clipboard. When the clipboard is used as a playalong score, each row of parameters is played for its specified duration, starting from the top row and proceeding in sequence down the list. Additionally, the currently playing parameter row is highlighted as it is played, to provide a visual cue to the score behavior. A playing score will repeat, causing continual changes in the controlled process's parameters, until the user stops it.

While a playalong score is playing, the user is able to start and stop recording training examples using the "Begin recording examples into dataset" button, the foot pedal controller, or OSC control messages, exactly as described above in Section 3.4.5.

We introduced playalong example recording in Fiebrink, Cook, and Trueman (2009). In that work, we discussed an extension of this playalong interaction, called

"playalong learning," in which the learning algorithms are continuously retrained on a background thread as a performer records new training data using the playalong interface. This functionality is not present in the current version of the Wekinator, whose GUI has been greatly redesigned since that publication. However, we plan to re-implement this functionality in the new GUI in the future, and until that occurs the older, playalong-learning-enabled Wekinator will remain available for download at `http://wekinator.cs.princeton.edu/`.

3.5 Other Features of the Software

3.5.1 Running Saved Models and Command-line Support

In order to run a trained model in a performance, it is possible to load a saved learning system from a file. The user may also opt to load the associated ChucK configuration and feature configuration from a file, bypassing all interactive configuration and setup steps. Furthermore, it is possible to run the Wekinator with command-line options to automatically load a ChucK configuration, feature configuration, and/or learning system, as well as to automatically run the learning system and minimize the Wekinator GUI as soon as the Wekinator launches. This allows a trained learning system to be applied in real-time to a problem without a user interacting with the Wekinator GUI at all. For example, a composer can save a learning system, ChucK configuration, and feature configuration for her piece, then distribute a simple command-line script to performers so that they may play the piece without ever using the Wekinator GUI.

3.5.2 OSC Control

Throughout this chapter, we have indicated that control over several aspects of the Wekinator state can be achieved using OSC control messages, in addition to the GUI. In summary, the set of Wekinator behaviors that may be controlled using OSC include: starting and stopping RECORDING, starting and stopping RUNNING, starting and stopping a playalong score, and initiating training of the models. These control messages may be used to implement an alternative user interface to the Wekinator, for example an interface embedded within a GUI designed for the performance of a specific composition. This feature also allows the Wekinator to be controlled remotely; for example, a "conductor" machine may multicast OSC messages to many other machines, triggering synchronous Wekinator behaviors across multiple performers.

OSC control over the Wekinator can be used simultaneously with user control via the GUI, and the user is able to enable or disable OSC control of the Wekinator at any time using a top-level menu item.

3.5.3 Compatibility with Weka

The Wekinator has been designed to maximize compatibility with Weka. The Wekinator can read and write training datasets saved in the Weka ARFF format. This

means that a dataset created interactively in the Wekinator could be loaded in the Weka GUI for experimentation with a wider variety of learning algorithms, feature selection methods, and visualization tools than are implemented in the Wekinator. Likewise, a training dataset created non-interactively (e.g., by running an audio feature extractor on a database of MP3s) can be loaded into the Wekinator, and a model trained on that data in the Wekinator can be run in real-time (provided there exists a way to extract identical features from real-time audio).

The Wekinator can also read and write trained models using Weka's method of Java serialization of Classifier objects. Therefore, a model trained in Weka can be imported into the Wekinator for running in real-time, even if was not created using one of the learning algorithms described in Section 3.3.3. Also, a model created in the Wekinator can be exported for use within any other Java project that uses the Weka library.

3.5.4 Example Feature Extractors and Controllable Processes

The Wekinator is distributed with several examples of custom ChucK and OSC feature extractors, ChucK synthesis classes, and OSC-controllable processes. Example custom ChucK feature extractors include: two keyboard feature extractors (one extracting the key row and column, and one buffering the last 10 key presses), a feature extractor for use with the Manta controller (Snyder 2010), two stereo audio feature extractors, and a vowel formant energy extractor. The Wekinator also comes with a simple spectral centroid feature extractor and a generic custom ChucK feature extractor code skeleton, to aid users beginning to write their own custom ChucK feature extractors.

Example OSC feature extractors include a simple three-slider GUI controller and two audio feature extractors, all implemented in Max/MSP. The audio feature extractors uses the analyzer~ object written by Jehan and Schoner (2001), and they computes features not included in the Wekinator's built-in audio feature set, such as estimated pitch, loudness, noisiness, and bark scale coefficients.

Example ChucK synthesis classes include: FM synthesis, audio live-sampling and panning, a bowed-string physical model (Smith 1986; Cook and Scavone 1999), a simple drum machine (the one described in Section 3.2.2), melodic synthesizers with continuous and discrete pitch control, and a simple "twinkle twinkle" melody player whose parameter dictates which phrase of the melody will be played next.

Example OSC processes include three versions of a simple animation written in Processing, with continuous and discrete parameters controlling hue and position, and a Max/MSP program for controlling the synthesis of another physical model, the blotar (Stiefel et al. 2004).

3.6 Related Work

We presented a broad overview of related work in supervised learning, HCI, and music in the previous chapter. In that chapter, we also drew on that related work to motivate the need for the Wekinator as a new tool for interactive supervised learning in real-time problem domains. Here, we specifically discuss how the design features and interactions supported by the Wekinator relate to particular existing tools in interactive supervised learning and music, and to threads of research in other domains, including robotics and speech.

3.6.1 Interactive Supervised Learning

In allowing iterative editing of the training data and retraining, the interactive supervised learning workflow supported by the Wekinator is similar to that proposed by Fails and Olsen (2003) and used in related work by Fogarty et al. (2008), Shilman et al. (2006), and others discussed in Section 2.4.1. Like those systems, the Wekinator uses training set editing as a way to leverage users' domain expertise in supervised learning problems for which users understand the supervised learning goal and are capable of changing the training data to steer the model toward that goal.

While those systems mentioned above presume that there exists a fixed set of unlabeled data available for training and testing, the Wekinator allows users to create new training examples on-the-fly. Additionally, the Wekinator allows users to evaluate trained models by presenting them with new, interactively-generated inputs. In this regard, the Wekinator allows interactions similar to two other tools that aim to build models from a user's real-time gestures: the Exemplar system for sensor-based interaction design (Hartmann et al. 2007) and the FlexiGesture system for personalizing the audio mapping of a pre-built new musical control interface (Merrill and Paradiso 2005).

We share the goals of this prior work of moving beyond a conventional supervised learning paradigm and engaging user interaction and expertise in appropriate and effective ways. However, as a general-purpose tool for interactive machine learning in real-time domains, the Wekinator offers many other user interactions that these single-purpose tools do not. It is flexible regarding the number, type, and meaning of input features and output parameters. It offers a choice among different general-purpose learning algorithms for classification and regression, and it allows users a much greater control over the machine learning problem: users can iteratively change the learning algorithms, algorithm parameters, and features, in addition to the training data. The Wekinator is not limited to a single learning problem or a single application domain; it can receive features from any real-time source and send model outputs to control any real-time process, using OSC messages. And, as discussed, the Wekinator offers several mechanisms for users to record training data, visualize features, visualize and edit training data, evaluate trained models, control the learning process, and import or export models and datasets to and from Weka.

3.6.2 Musical Audio and Gesture

As discussed in the previous chapter, one motivation for creating the Wekinator was to produce a supervised learning tool for musical applications that—unlike existing tools—was general-purpose, capable of working on real-time signals, and supported appropriate user interactions. Table 3.2 uses these criteria to compare the Wekinator to several existing tools for applying machine learning in music: Weka (Hall et al. 2009), jMIR (McKay and Fujinaga 2009), Marsyas (Tzanetakis and Cook 2000), MarsyasX (Teixeira et al. 2008), MnM (Bevilacqua et al. 2005), Matlab with MIRToolbox (Lartillot and Toiviainen 2007) and MIDIToolbox (Eerola and Toiviainen 2004), the PD gesture library of Cont et al. (2004), and SMIRK (Fiebrink et al. 2008). This table does not include tools that have been created for a specific purpose, for example creating mappings for a specific input device, like work by Merrill and Paradiso (2005), or controlling specific synthesis algorithm, as in work by Brent (2010).

The Wekinator was also very much inspired by the collection of past work that has applied machine learning algorithms to particular musical applications, particularly work by Lee et al. (1991) and Fels and Hinton (1995). We refer the reader to Section 2.2.3 in the previous chapter for a more thorough overview of related research in this area.

3.6.3 Robotics and Programming by Demonstration

The Wekinator's approach to supporting end-user systems-building by demonstrating examples of the concept to be learned has parallels to prior work in programming by demonstration (PbD), also called programming by example. In PbD systems, the computer learns to perform a task by observing a human performing it. In *Watch What I Do* (1993, 1), Allen Cypher writes, "The motivation behind Programming by Demonstration is simple and compelling: if a user knows how to perform a task on the computer, that should be sufficient to create a program to perform the task." PbD has been used in robotics since the 1980s, as a way for users to more efficiently program robots used in automation processes (Billard et al. 2008). While PbD can be useful for programming rote execution of simple tasks (or creating "macros on steroids," in the words of Lieberman (2001, 2), machine learning techniques are also useful in allowing the program to generalize to scenarios that may differ from those demonstrated by the user. Machine learning algorithms have been employed in PbD applications in both robotics (e.g., Münch et al. 1994) and software (e.g., Lau et al. 2001).

The Wekinator can be viewed as a tool for applying machine learning to end-user PbD, in that users can *program* system behaviors (e.g., sounds played) to respond to particular user actions (e.g., hand motions in front of a camera). Again, its general-purpose nature (several algorithms, ability to use arbitrary input features and control arbitrary processes) distinguishes it from other tools in this space, and its real-time and interactive capabilities allow it to be applied to real-time user demonstrations for which conventional machine learning tools like Weka and Matlab are inappropriate.

Table 3.2: A comparison of the Wekinator with other tools used for supervised learning in music. Columns include: name, whether it is can run on real-time input signals, whether using the tool requires the writing of code, whether it explicitly supports interactive editing of the training data (excluding the obvious ability for end-user developers to add their own interfaces to do this, for tools that are libraries or frameworks), the built-in learning algorithms, the application domain, and the environment in which the tool is run, if any.

Name	**Real-time**	**Requires coding**	**Interactive data editing**	**Built-in algorithms**	**Domain**	**Environment**
Weka GUI	No	No	No	Many	Any	
Weka Library	Yes	Yes	No	Many	Any	Java applications
jMir	No	No	No	Many	Audio, MIDI, text	
Marsyas	Yes	Yes	No	knn, SVM, Gaussian	audio, MIDI	C++ or other code w/ bindings
MarsyasX	Yes	Yes	No	knn, SVM, Gaussian, clustering	audio, gesture, other	C++ or other code w/ bindings
MnM	Yes	No	Yes	Matrix	Gesture	Max/MSP
MIR Toolbox, MIDI Toolbox	No	Yes	No	None (use within Matlab)	Audio, MIDI	Matlab
Cont et al. (2004)	Yes	No	No	Neural networks	Gesture mapping	Pd
SmirK	Yes	Yes	Yes	kNN, AdaBoost, J48	Audio, gesture	ChucK
Wekinator	Yes	No	Yes	kNN, AdaBoost, J48, SVM, Neural networks	Audio, gesture, real-time applications using OSC	

However, as we will see in the next chapters, to view the Wekinator primarily as a PbD tool is inappropriately limiting. First, the goal of using the Wekinator is not to create a model that replaces a human in performing automated tasks, but to create models that will be useful in some interactive context. Additionally, the context for application of the trained models is different, in that humans (possibly the same humans creating the models) may be both designing and participating in these interactive contexts. Furthermore, the Wekinator can also be used in applications where the user does not have clear a priori ideas of how the trained models should behave; the aim of the learning problem might not at all resemble the human user attempting to teach the computer about a concept with which he is familiar.

The playalong example recording process also invites parallels to PbD, in that it generates training data as a human user demonstrates an action to the computer in real-time. However, in conventional PbD, the human user is demonstrating to the computer the actions that *the computer* should emulate under a given input state, while in playalong recording, the human is demonstrating the *human* actions that should ultimately result in a given computer-generated behavior. In playalong recording, the formulation of the machine learning problem is actually reversed from both PbD and the "active learning" (Lewis and Gale 1994) approach to engaging user interaction with machine learning, in that here the computer is prompting the user to provide a feature vector for a given model output, rather than prompting the user to provide the desired output label for a given feature vector.

3.6.4 Speech Recognition

In this regard, playalong example recording is more akin to the training interfaces employed by speech recognition systems, in which a user is asked to speak a sentence or series of words presented to her by the computer. Speech recognition systems have employed these types of user prompts to allow users to create and improve speaker-dependent recognizers since at least the work of Jelinek et al. (1975). Such interfaces are now common in commercial software such as Dragon NaturallySpeaking (Nuance Communications, Inc. 2010), as well as speech recognition built into the Windows operating system (Microsoft Corporation 2010).

When the goal of supervised learning is to model behaviors of the user himself, as is the case in both these speech recognition systems and in many Wekinator applications, it is appropriate to rely on the user to provide additional information to improve the model's performance. Prompting the user for new training examples is a straightforward way of obtaining such information. Ongoing research on other mechanisms and interfaces for effectively leveraging user interaction in speech recognition systems, such as work by Vertanen (2009), is therefore relevant to the goals of the Wekinator. Of course, the Wekinator may be applied to a broad array of applications, whose characteristics regarding the difficulty of the learning problem, the extent of the user's flexibility in adapting her behavior in interacting with the system, and the ease with which new training data may be provided may be very different from each other and from other applications, including speech recognition. We will discuss these

differences and their implications for interface and algorithm design in later chapters, as well.

3.7 Conclusions

The Wekinator is a general-purpose tool for applying standard supervised learning algorithms—both classifiers and neural networks—to real-time problem domains. It leverages human interaction in creating and modifying the training set and evaluating trained models in real-time, as well as in modifying learning algorithms and features and computing standard evaluation metrics. It is tailored to musical applications: it incorporates built-in feature extractors for audio and gesture signals, as well as a set of example custom feature extractors and musical processes meant to help composers and musicians get started with the application. At the same time, it can be applied to arbitrary learning problems outside music, using feature extractors and controlled processes that communicate with the Wekinator using the OSC protocol.

While many improvements could be made to the Wekinator, including support for more algorithms, a wider variety of built-in feature extractors, and additional interfaces to aid the user in visualizing, understanding, and modifying aspects of the learning problem, the current version of the software meets our goals described in the previous chapter for a general-purpose supervised learning appropriate for use in music.

In the following chapters, we discuss how the Wekinator has been applied to problems in music composition and performance, and how our experiences working with users have led to the inclusion of some of the design features described above.

Chapter 4

Participatory Design Process with Composers

4.1 Introduction

In this chapter, we discuss our work in Autumn 2009, in which we collaborated with seven composers as they learned how to use the Wekinator, employed it in the creation of new musical instruments and compositions, and contributed to the improvement and evolution of the software. The collaboration was structured as a participatory design process, in which composers provided feedback on the Wekinator and proposed new features or improvements over several development iterations.

Several key features of the Wekinator described in the previous chapter grew out of this collaboration, and throughout this process, the Wekinator improved with regard to the breadth of applications it supported, the scope of user interactions it enabled, and the robustness of its user interfaces and underlying implementation to use by a variety of users with different goals and backgrounds. Furthermore, by observing composers using and discussing the Wekinator, we gained a deeper understanding of how the Wekinator supports their creative priorities and goals, and how it influences and informs their work. This research has thus enabled us to contribute to the still small body of work studying human-computer interaction in computer music composition and instrument design, and our findings underscore our belief that further research in this area will yield insights that are important to both composition and HCI.

We begin this chapter with a brief discussion of relevant prior work on HCI in composition and instrument building, an introduction to participatory design as used in software design and HCI research, and details of the method employed in this work. We discuss how the participatory design process unfolded in practice, our observations of how composers used the Wekinator and the interaction strategies they employed to accomplish their goals, the qualities of the Wekinator that composers found most valuable, and the changes made to the Wekinator throughout the participatory design process. Subsequently, we present our evaluation of the Wekinator as a compositional tool, offer a new perspective on the dichotomy between generative

and explicit mapping strategies, and present initial discussion on the way that the Wekinator influenced composers' practices. We further discuss implications of this work in the contexts of interactive supervised learning and computational creativity support in Chapters 8 and 9.

The key findings of the work in this chapter are as follows: Composers found the Wekinator to be a valuable tool that allowed them to create mappings for new musical instruments and interfaces that were more expressive than using other techniques, and to create them more easily than other techniques. They developed interactive strategies for working with the Wekinator that involved using the training data to initially sketch out their ideas, evaluating trained models in a hands-on manner, and iteratively modifying the training data to change the mappings. They liked the speed and ease with which the Wekinator allowed them to create and explore mappings, the way the Wekinator allowed them to privilege the gesture-sound relationship via physicality and abstraction, the ability to use the Wekinator to access surprise and complexity, and the ability to balance surprise and complexity with predictability and control. The many improvements made to the Wekinator software during this process centered around supporting greater control and constraint over the learning problem, better enabling composers to take advantage of discoveries made using the software, further supporting abstraction and physicality, improving usability for diverse users, improving compatibility with composers' preferred tools, providing more information about the learning state, and reducing barriers to rapid prototyping and exploration.

4.2 Background

4.2.1 Motivation in the Context of Prior Work in HCI and Composition

One of the great opportunities presented by computer music is that the mechanisms through which humans produce and control sound during a performance are flexible and open to design and change by a composer. In composing music for acoustic ensembles, the role of the composer may be limited to the specification of the sounds each performer is to produce over time; the performer's physical actions and the relationships between physical gesture and the resulting sound are circumscribed according to the acoustics of the instrument and and by instrumental performance techniques. In computer music, on the other hand, a composer is free to design new physical interactions between humans and their instruments, and new relationships between these actions and sound. His design choices affect everything from the sounds that can be produced to the effort and practice required of the performers, the techniques by which performers may exercise expressive agency, and the audience's perceptions of the role of humans and technology and of the structure and meaning of the piece. In short, whether these design choices are conscious or not, and whether they are dictated by aesthetic or practical concerns, they are part of the composition. For these reasons, computer music interfaces in which performer action is not physically coupled to sound production have been called "composed instruments" (Schnell and

Battier 2002), and we—like the composers with whom we worked—will not make a clear distinction between "composition" and "instrument design" in this chapter.

There exists a body of research applying an HCI perspective to studying computer music, which primarily focuses on the performance-time interactions between a performer and a musical interface or "composed instrument." For example, prior work by by Wanderley and Orio (2002) and Hunt et al. (2002), has studied the efficacy of different types of mapping functions from control interfaces to sound synthesis parameters (see the discussion on mapping in Section 2.2.3).

While that work is informative regarding the practical and musical consequences of different potential manifestations of a computer music instrument, it says nothing about the *process* by which composers design new instruments and performance-time interactions. This process is itself mediated by technology, as composers engage with hardware and software tools to create new instruments and compositions. Composers' and instrument designers' writings reflecting on their own experiences, such as work by Hahn and Bahn (2003) and Cook (2001), offer valuable insights into the practice of composing the instrument and the ways that the tools of composition (including software, hardware, and algorithms) influence their working processes. A broader analysis of "interaction" (including human-computer interaction) in composition can be found in Paine (2009), including an analysis informed by the role of embodied cognition in interactive system design (Van Nort 2009), and by an interrogation of the definition and scope of interaction in music (Drummond 2009).

The work presented in this chapter seeks to provide a complimentary perspective to this prior work. We have approached questions regarding how to design technology for use in composition and instrument building, and how technology influences the composition process, through a methodology informed by HCI. By working closely with several composers using the Wekinator in their work, and by directly involving them in the design of the Wekinator itself, we hoped to better understand what aspects of the Wekinator composers most valued as a compositional tool, and how various interactions with it supported their creative work in composition and instrument building. This understanding helped us both to improve the Wekinator software and to further contribute to an HCI perspective on how new software systems can support the work of computer music composers and instrument builders.

4.2.2 Participatory Design

We have applied a participatory design approach to our collaboration with composers. Participatory design, also called user-centered design, describes a process of engaging users of a software artifact throughout the design and implementation process. Originating in the work of Norman and Draper (1986), user-centered design is described by Vredenburg et al. as including "the active involvement of users for a clear understanding of user and task requirements, iterative design and evaluation, and a multi-disciplinary approach." This practice is used in product development within commercial IT companies, for example at IBM, to create products that are more usable by and useful to their intended users (Vredenburg et al. 2002; Vredenburg 1999).

Participatory design has also been used in HCI research, including in the study of music composition. Recent work by Tsandilas et al. (2009) employs an participatory design approach to studying a new digital pen-based score annotation tool for music composition. The authors studied five composers interacting with the tool, and they involved the composers in a participatory design process to iteratively improve the tool. The outcomes of the study included both the implementation of new software functionality and an improved understanding of the roles of paper and computers in composers' creative processes.

As we discuss below, the primary aspects of participatory design that we have employed in our work include: observing how composers used the software in practice to perform their work, soliciting their ideas for new features that would enable them to use the software more effectively or in new ways, regularly presenting composers with iterative re-designs of the Wekinator that implemented new interaction techniques and interfaces, continually seeking composers' feedback on each iteration of the software and the accompanying documentation, and providing tools to facilitate group communication in between face-to-face meetings.

4.3 Method

We worked with seven composers associated with the Music Composition program at Princeton. Participants met weekly for ten weeks in the fall semester of 2009. The Wekinator software, which had been undergoing development during the previous year, was a focal point of the meetings and a foundation for the collaborative study: composers were asked to learn to use it, employ it however they found it to be most interesting as a compositional tool, and critique, improve, and test iterative revisions of the software. The participation of the composers was not otherwise constrained, and they were not obligated to produce a composition, instrument, or other artifact, nor were they graded or otherwise compensated. Most composers were intrinsically motivated to participate by their curiosity to learn about machine learning and explore whether and how it might be useful to their work; over the course of the design process, some composers were also motivated by a desire to incorporate machine learning into pieces they were working on.

Participation was open and advertised to all composers at Princeton. Of the seven who chose to participate, six were PhD students and one was a faculty member; four were male and three were female. Their self-assessed programming expertise ranged from 1 ("very low") to 4 ("somewhat high") on a 5-point Likert-style scale (mean = 2.9). Six rated their prior machine learning knowledge as "very low" (1) and the seventh rated it as "somewhat low" (2). None had used machine learning or generative mapping strategies before in their compositions, for any purpose, though most had experience designing and using explicit mapping strategies. Composers all had prior experience creating computer music compositions using ChucK and/or Max/MSP.

All meetings involved group discussion and time for individuals to experiment with the Wekinator, share their work with each other, and solicit feedback and help from the group. We asked them to share feature requests and bug reports, invited group

discussion of proposed improvements, and noted which aspects of the software caused people difficulties. Each week, we implemented the requested functionality into the Wekinator (as was feasible) and distributed an updated version to the composers before the following meeting.

During each meeting, we recorded text minutes of the group's activities, discussion topics, questions, problem reports, and requests for changes to the Wekinator's user interface or functionality. The group communicated via an e-mail list, and composers sometimes e-mailed and met with us directly. After ten meetings, each composer completed an individual electronic questionnaire soliciting reflective feedback and demographic information using 20 multi-part free response and Likert-style questions. Additionally, the Wekinator's version control system (SVN) contains a record of all the changes that were made to the Wekinator during this process. This set of documentation forms the basis for the following discussion of the participatory design process outcomes, composers' use and evaluations of the software, and the improvements made to the software during this work.

4.4 Outcomes

4.4.1 The Participatory Design Process in Practice

While there was no officially designated meeting length, most weekly meetings lasted around three hours, and this time included both discussion and hands-on experimentation. Discussion topics in the first few weeks of the process centered around teaching composers how to use the current version of the Wekinator software, brainstorming ideas for how they might use it in new projects, and formulating a set of shared goals for the group. Composers' goals for the semester included learning about the Wekinator, experimenting with machine learning, discovering techniques or ideas that could be used in future compositions, building instruments that they could use in performance, and collaborating on future academic publications pertaining to the Wekinator and its use in composition.

In the second week, two composers shared new instruments that they had begun to create with the Wekinator, and a third spent time in the meeting trying out his ideas for a new project. By this time, several composers had enough experience working with the software that they began to make suggestions for how it might be improved and modified to be more easily applied to the projects that most interested them. The first design iteration was thus begun, and a new version of the Wekinator was distributed to the composers before the third meeting. In all, eight such weekly design iterations were completed, though many ideas from composers have continued to be integrated into the Wekinator since the end of the participatory design process.

From the third week on, meetings typically started with a discussion of how the Wekinator had been changed over the last week. When changes were major, we used an overhead projector to demonstrate to the composers how to use the new features. During the discussion period, composers offered their feedback on the changes to the software and discussed how they had been using the Wekinator over the past week.

Composers often had suggestions for new software features related to how they had been using the Wekinator, or how they discovered they wanted to be able to use the Wekinator. These suggestions were typically discussed as a group, and we provided our own input regarding what would be feasible to implement over the short- and long-term.

Most suggestions and feedback centered around the user interactions that were possible with the Wekinator, including the workflow supported in the GUI, the extent to which users had control over and feedback about various aspects of the machine learning process, and the ease with which the Wekinator could be used in conjunction with other software and by people who were less comfortable programming or using a command-line interface. Discussions of the terminology used in the documentation and user interface and of the visual presentation of the interface itself were less frequent, though significant.

Another portion of every meeting was devoted to hands-on experimentation. In this period, composers usually worked individually, on their own laptops, to try the updated version of the software for themselves and work on their own projects. Periods of working independently were frequently punctuated by composers sharing their accomplishments or discoveries with the rest of the group, for example demonstrating a mapping that they had just created and encouraging others to play with it.

The group communicated between meetings using an email list, a wiki[1], and in the last two weeks, a Google Wave group. The wiki was mostly used to share a schedule of the group's activities, resources for people who wanted to learn more on their own about machine learning, instructions for updating and running the Wekinator software, and a running list of bug reports and feature requests for both the Wekinator software and its documentation. While we encouraged composers to edit the wiki themselves to add their ideas or questions, no one did; instead, they opted to share ideas and questions verbally during meetings or over email. Composers used the email list fairly regularly for discussions, questions, and sharing feature extractor and synthesis code with one another. They posted 28 messages in total over the eight weeks between the email list's creation and the end of the ten-week design process. (This count does not include announcements we made to the email list or posts we made answering people's questions.)

During both the experimentation portions of meetings and through email correspondence on the group mailing list, composers often sought help from each other and from us when they did not understand some aspect of the Wekinator or were having trouble accomplishing what they wanted to do. While we attempted to help people accomplish their goals by using the current version of the software more effectively, we also sought to understand the underlying causes of their misunderstandings and difficulties; discovering barriers to the Wekinator's usability informed many of the changes that we made to the software.

By the end of the process, most composers had become very familiar with the Wekinator and were able to use it effectively. Many of their questions therefore turned from asking us how to accomplish certain tasks with the software, or whether

[1] A copy is available at `http://www.cs.princeton.edu/~fiebrink/thesis/resources.html`.

certain tasks would be possible, to asking us about the underlying machine learning algorithms and principles. In the ninth week, we led a high-level discussion about how the machine learning algorithms in the Wekinator worked.

After the tenth meeting, coinciding with the end of the semester, composers completed the final questionnaire. Over the following several weeks, we collaborated on the preparation of a conference paper about the Wekinator and our use of the participatory design process to study human-computer interaction in composition, which was presented at the 2010 International Computer Music Conference (Fiebrink, Trueman, et al. 2010); much of the content of that paper is discussed in this chapter. Over the following months, two of the composers developed the projects they had been working on with the Wekinator into components of pieces that were publicly performed. We discuss those pieces in detail in Chapter 7.

4.4.2 How Composers Used the Wekinator

A Focus on Building Continuous Gestural Controllers

The majority of composers' work with the Wekinator focused on using neural networks to build new interfaces for continuous gestural control of sound synthesis, where the input features were extracted from a gestural controller of some sort, and each neural network output drove a parameter of a sound synthesis algorithm. The most commonly used input devices included the laptop accelerometer, the GameTrak Real World Golf USB controller (shown in Figure 4.1), and the 3Dconnexion brand SpaceNavigator USB controller (shown in Figure 4.2). (The choice of the GameTrak controller was inspired by its use in *Pendaphonics* by Skriver Hansen et al. 2009.) Other input devices used included Wacom tablets, micro-controllers outfitted with sensors, laptop trackpads, webcams, keyboards, and USB joysticks.

Most of these devices, including the GameTrak, SpaceNavigator, and joysticks, had been previously purchased for use in projects within the Music Department, and were available for any participant to use. Others, including the Wacom tablet and micro-controllers, were owned by participants themselves, who brought them to the meetings to experiment with them. All participants experimented regularly with at least two input devices over the ten weeks, and most experimented with three or more devices.

The synthesis algorithms most commonly used by composers were the "blotar" and "uBlotar" algorithms. Both were implemented as Max/MSP objects (Stiefel et al. 2004) and controlled by the Wekinator via OSC. These algorithms produce sound using a physical model of the reverberations of a flute and electric guitar. Capable of producing both flute-like and guitar-like sounds, they include twelve or more real-valued parameters having complex and interdependent effects on the sound produced by the algorithm. (Because of this, Stiefel et al. describe the blotar as a"remarkably difficult instrument to control and perform.") The next most popular synthesis algorithms used were granular sampling and synthesis, which were implemented within ChucK synthesis classes.

Figure 4.1: The GameTrak Real World Golf USB controller. The base unit houses two spring-loaded strings that can be pulled out from the base and moved in 3D space. The unit can be employed as a 6-axis HID device (x-, y-, and z-axes for each string). The controller also includes a detachable button that can be pressed using the foot.

A few composers did experiment with discrete classifiers and with using audio input features during the first few meetings, but they did not ultimately pursue these projects and ended up working with gestural inputs and neural networks for the rest of the ten weeks. We discuss in Section 4.4.6 some reasons why it was less practical to use audio inputs, and we discuss throughout Section 4.4.4 several reasons that composers enjoyed using the neural networks.

Interactive, Iterative Application of Supervised Learning

Composers used the Wekinator to quickly prototype and experiment with different mappings from gesture to sound, and they used the Wekinator's interactive machine learning capabilities to iteratively modify models after their creation. Five of the seven composers responded to all the final questionnaire items inquiring about the steps they took to modify the neural networks' behavior when they were not satisfied with the mappings. (The two composers who did not respond had used the Wekinator the least among the group; we discuss their participation in Section 4.5.1.) Figure 4.3a shows the number of composers out of these five who had, at some point, tried each action to attempt to improve a trained model, and Figure 4.3b shows how the composers who had tried each action rated its usefulness. Based on these outcomes, the most useful methods for improving mappings were adding more examples to an

Figure 4.2: The 3DConnexion brand SpaceNavigator USB controller. The knob, designed for navigating through 3D environments, can be moved laterally from left to right and front to back, pushed and pulled vertically, tilted from side to side and from front to back, and twisted around its center axis, yielding six features describing its position.

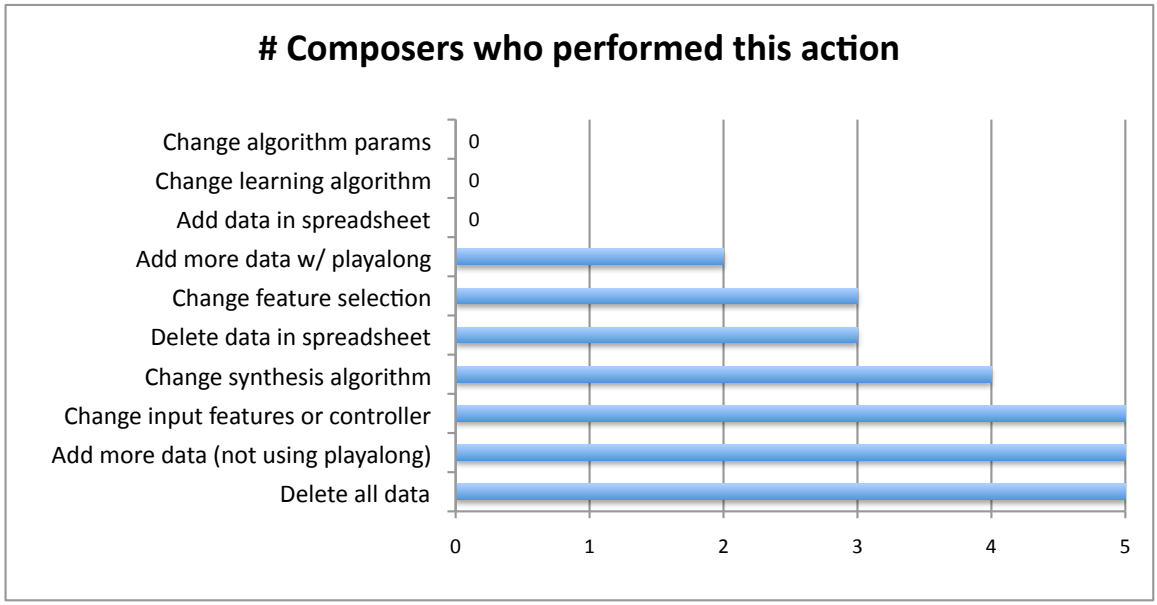
existing training set (5 had tried; mean usefulness score 2.6 out of 3), deleting the training set and building a new training set from scratch (5 tried; mean score 2.8), changing the input features or controller (5 tried; mean score 2.4), and changing the synthesis method (4 tried; mean score 2.4). The two composers who had used playalong example recording both rated adding examples using playalong as highly useful. Composers rarely or never added training data manually or modified the neural network architecture or training parameters, though these tasks were possible. In summary, it is clear that composers found it most useful to improve models by iteratively changing the the training data rather than by taking actions supported by conventional machine learning tools, such as modifying the learning algorithm and features.

4.4.3 Strategies for Using the Wekinator

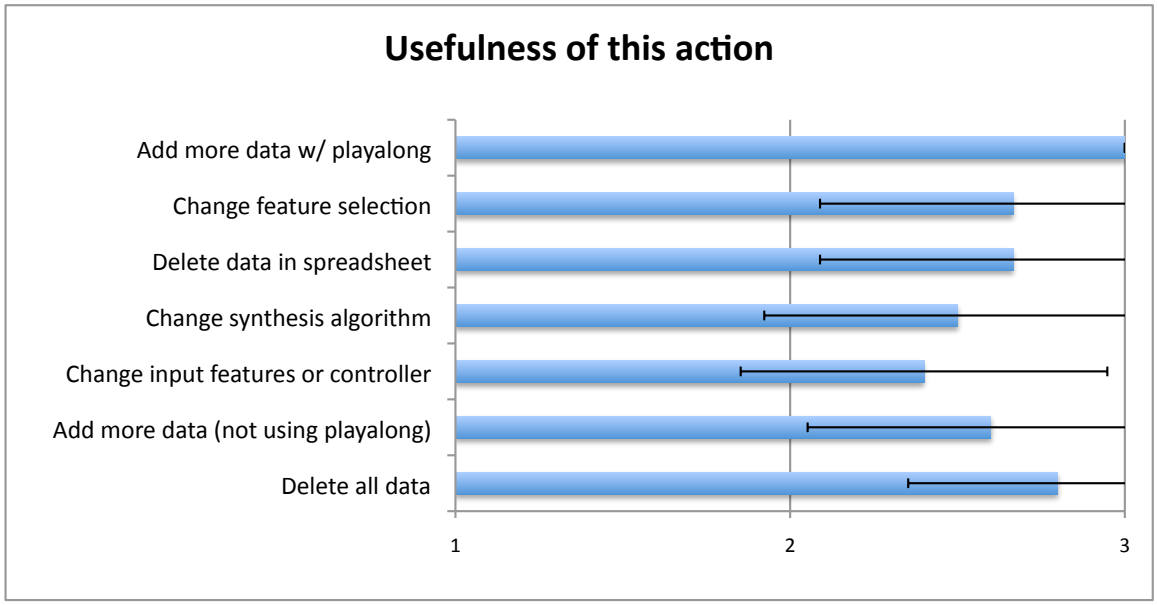
Training Data as a Sketch

One of the five composers who responded in detail to the questionnaire prompt regarding the strategy for creating new mappings with the Wekinator indicated that he sometimes had quite specific gesture-to-sound mappings in mind that he worked to teach the Wekinator. The four others used the Wekinator to build mappings from less well-formed ideas; for example, one wrote, "I typically find some synth presets [i.e., parameter settings] that I like, record a few examples, and train the system. So far I have not recorded very many examples, but rather I explore the space the training sets up. In my experiments so far I haven't really had a preconception of how I want the mapping to work; I've been content to discover things about the mapping the Wekinator came up with."

This composer's strategy is typical of this latter set of four composers, in that they all used the training set as a sort of "sketch," roughly mapping out their ideas about



(a) The number of composers who had used each action, out of the five who replied to these questionnaire items.



(b) The usefulness of each action, rated by composers who had used it, where 1 = "not at all useful," 2 = "somewhat useful," and 3 = "very useful."

Figure 4.3: The use and rating of actions for modifying and improving trained models.

the set of sounds and gestures that they knew they wanted to include in the model. They then relied on the neural network training to preserve the behavior sketched out in the training data with some faithfulness, while also filling in the rest of the mapping behavior in some useful (and perhaps unexpected) way. While most composers did not usually employ playalong example recording, the composer who used playalong the most also employed a similar strategy: "I usually carefully pick 4–6 extremes in the synthesis algorithms, places I'd like to be able to reach in some way. Then I set up a playalong score that cycles through these, and I practice, sometimes at length, playing the controller along with the score. Then I play along, maybe several times, sometimes repeating the same gestures, sometimes usually different—usually extreme—gestures that explore different parts of the interface."

Certain mapping qualities were especially valued by composers; most notably, several composers explicitly added examples to ensure that the mapping would include a gesture that reliably produced silence. One composer who was particularly open to exploring whatever mappings the Wekinator might create from his ad hoc training data ("I've been content so far to discover what the wekinator has come up with") still always began by adding "examples where the synth is not making any sound. Then I add a few other examples with different configurations and sounds and train. Not a whole lot of strategy!"

Certain composers developed strategies in which they explicitly used the training examples to not just denote points of interest in the sonic and gesture spaces—sounds and gestures they knew they wanted to have available in the mapping—but also to denote and control boundaries of the sonic and gesture spaces. For example, one composer wrote, "I aim to combine extremes with extremes, mostly—take an extreme controller position, connect it an extreme synthesis setting." Another wrote, "I keep adding trainings. . . until it feels like I've covered the outer and inner limits of my gestural landscape. I don't spend much time training the mid-range of gestures. Not sure why, just haven't done that."

Hands-on Evaluation

Once an initial mapping was created, all composers evaluated it by running it on new gestural inputs in real-time. That is, they played the new instrument they had just created. In their playing, composers indicated that they tested the mapping against several criteria. One criterion was the extent to which the mapping faithfully modeled the training examples, for example: ". . . I would make sure that I could easily recreate the various positions at which I had trained the model." However, beyond faithfulness to the training data, composers' mapping evaluation criteria were quite subjective and informed by their musical expertise. Responses included: "I play with it to see if I like how it sounds," "The mapping should be capable of subtlety," "I like when gestures feel like they 'match' a sound," and "If I feel a certain level of reproducibility, predictability, and a balance of consistency and surprise, it seems to make me happy." Additionally, a few composers indicated that they expended some time and effort practicing playing the mappings in order to better evaluate them against these criteria: "It takes time to learn what the controller/mapping does."

No composers ever used cross-validation or training accuracy metrics to evaluate their models, for reasons that likely included both the fact that they were not familiar with the metrics, and the fact that the subjective criteria above had little to do with notions of accuracy or generalization. We present further discussion of the meaning of supervised learning evaluation metrics in interactive and creative contexts in Section 8.5.2.

Improving the Mappings by Adding and Deleting Data

As discussed above in Section 4.4.2, the actions composers found most useful for improving trained models against their evaluation criteria were adding more examples to an existing training set or deleting the training dataset and adding new examples from scratch.

One reason that composers added training data was to reinforce a behavior that had not been learned correctly on the previous iteration of the training set. For example, the composer whose training data creation always began with adding examples that produced silence wrote, "Typically the only times I add new training examples are when the initial [mapping] doesn't have a predictable place where it's silent."

Composers also commonly added training data not to correct a model's behavior, but to further develop a mapping whose initial behavior seemed promising. One composer wrote, "If I like [the mapping], I will save and go on and start adding new examples." Another wrote that he added data to enforce specific, predictable behaviors in portions of the mapping space: "I try to add additional points in between the extremes where I know what I want to be most predictable... These additional points were my own imagined 'just right' between two such extremes."

Three composers indicated that they most commonly deleted all training data and started over from scratch when they were dissatisfied with how a mapping performed. Their responses suggest that this is due to the fact that simply starting over and providing a new set of better training examples involved a relatively low overhead of time and effort. Composers also used the deleting of individual "recording rounds" (i.e., training examples recorded during the same instance of the Wekinator entering a RECORDING state; see Section 3.4.5 on page 66) to undo changes to a mapping when they discovered their modifications had not had a beneficial effect. One composer indicated that he was likely to delete only a subset of the training examples if he had already spent some time refining the mapping; if he was early in the mapping creation process, he was more likely to delete everything and start over.

One composer applied a more fine-grained strategy for choosing training data to delete, including searching for and deleting outliers and pruning his dataset to include fewer examples of each class: "having too much data seemed to cause the model to even everything out to a gray middle ground, so nothing would change. I had about 60 examples of each data point (i.e. holding the controller in one position, and taking data for 5 seconds). I deleted all but 10 of each of these and it worked much better."

4.4.4 What Composers Liked About the Wekinator

Speed and Ease of Creating and Exploring Mappings

When asked what aspect of the Wekinator was most useful to their work, four of the composers' responses included mention of the speed with which the Wekinator allowed them to create and experiment with new mappings. Several composers expressed frustration at the difficulty and slowness of creating and changing mappings using previous tools; one composer who has been building new musical interfaces for over ten years wrote, "Building these kinds of instruments requires an enormous amount of experimentation and play. There are simply way too many combinations of features and parameters to manually think about trying—too many decisions to make—and too many combinations that are useless. It's a process that invariably takes way too much time; the ratio of time spent to satisfactory musical experiences is super high." In contrast, composers frequently referenced the speed and ease of creating and exploring mappings using the Wekinator. One composer wrote, "As I work mostly with improvisation, I found Wekinator's ability to quickly map a number of input features to a potentially different number of output parameters very useful." Another wrote, "What I've enjoyed most about the Wekinator to this point is the speed and ease with which controllers can be prototyped, and different mappings experimented with."

Privileging the Gesture-Sound Relationship via Physicality and Abstraction

When asked what they liked about creating mappings with the Wekinator and how they felt it was useful to their compositional practice, a common thread throughout composers' responses was the privileging of the relationship between gesture and sound, both in composition and performance. One obvious way in which the Wekinator emphasizes an embodied approach to composition is by allowing the user to generate training examples by actually performing gestures in real-time, as opposed to approaching the mapping creation process rationally or mathematically, as is required in the design of explicit mapping functions (see Section 2.2.3 for a discussion of explicit and generative mapping techniques). Several people contrasted their experiences with the Wekinator with their previous experiences creating instruments that used explicit mappings coded in Max/MSP or Chuck. One wrote, "I find that I've been so overwhelmingly trained to think of these things in one-to-one ways (uh, I want tilting forward to control volume, and, uh, tilting left/right to, uh, control.... pitch, yeah that's it) that I basically want to retrain myself NOT to think that way anymore, and rather to privilege physical interactions with the interface and sonic sculpting on the synthesis end, and ask the Wekinator to connect them for me."

Composers frequently discussed the Wekinator's abstraction of the mapping functions as being useful in focusing their attention on sound and gesture, and contrasted this to explicit mapping methods that drew their attention and effort to refining the details of the mapping functions that had minor or unpredictable relationships to the sound and movement. One wrote, "By thinking about the mapping mechanism... as

a closed 'black box', Wekinator allowed me to explore the parameter space in a more intuitive and natural way. In the past when I have really focused on trying to create an expressive mapping between inputs and outputs, I ended up spending so much time and energy on the mappings themselves that they started to eclipse the actual sonic result."

Several composers also commented on a prioritization of the gesture-sound relationship in the evaluation and performance of mappings after they were created. In discussing how they evaluated a given mapping, all users emphasized the importance of playing with it and getting a "feel" for it. This "feel" had particular implications for performance and composition: "They (the performers) can see a relationship between my control gestures and the sounds. For dancers, at least the ones I work with, this is really important as they want to have a sense of the musician being in it with them, 'getting' the movement, rather than just playing back canned sounds that have no relationship to the movement of the moment." One person reflected on the type of composition suggested by his use with the software: "I could imagine using the Wekinator for a piece where the instrument itself was the focal point, possibly some kind of concerto, or where there is an element of theatrics to the performance. I think that the gestural relationships that the Wekinator creates between the controller and the sound producer [e.g., synthesis algorithm] would be very interesting to highlight in a piece."

Access to Surprise and Discovery

Composers strongly emphasized the creative benefits of being surprised by the sounds and by the sonic-physical gestures that resulted from the generated mappings. For example: "It's... nice to find new sounds that I hadn't anticipated," and "There is the potential of creating very expressive instruments using mappings from the Wekinator. In traditional circumstances (before the Wekinator) programmers might try to constrain the values in such a way that they're always getting the results they wanted for whatever piece the instrument was being designed. Nothing wrong with that. But with Wekinator it's possible to get the controller into an unanticipated configuration and get an unexpected sound. Such a sound might not have a place in the piece the composer is working on, but might if that instrument is used in another piece." Another wrote, "There is simply no way I would be able to manually create the mappings that the Wekinator comes up with; being able to playfully explore a space that I've roughly mapped out, but that the Wekinator has provided the detail for, is inspiring." The reliance on neural networks to create something novel and even unexpected from the training examples contrasts sharply from conventional supervised learning applications, as we discuss further in Chapter 9.

Access to Complexity

Almost all of the composers wrote that they valued the ease with which complex mappings could be created using the Wekinator. Two types of complexity were mentioned as being important: first, composers valued being able to easily construct mappings

in which a single gestural input feature affected multiple sound parameters, and in which each sound parameter was affected by multiple input features. That is, it was important to be able to create many-to-many mappings, as opposed to mappings in which each feature affected a single parameter. Second, composers valued being able to easily construct mappings in which output sound parameters were computed as complex functions of the input gesture features, as opposed to using linear, exponential, or other simple mathematical functions.

These types of complexity are more easily achieved using the Wekinator than the explicit mapping techniques composers had previously employed. Whereas the effort required to explicitly craft a mapping function is related to the function's number of inputs and outputs, the Wekinator's example-driven, generative mapping paradigm incurred the same amount of work for creating many-to-many mappings as one-to-one mappings. Furthermore, the Wekinator's neural networks were capable of producing functions with highly non-linear, complex behavior.

The musical value of many-to-many mappings in computer music interfaces has been noted in prior work by Hunt and Wanderley (2002). Furthermore, composers' value on complexity is understandable in the context of musical practice: in playing an acoustic musical instrument, a performer's gestures affect the perceived qualities of the instrument's sound in a highly nonlinear way, and with a many-to-many relationship. These properties are responsible for both the difficulty inherent in learning to play an instrument, and in the range of expression achievable by an expert performer. Several composers actually noted parallels between the degree of complexity and difficulty in Wekinator-created mappings and those in acoustic instruments, and they indicated that such parallels were a very positive phenomenon and not easily achieved using other mapping techniques. For example, one composer wrote, "Like any good instrument, acoustic or electronic, when the Wekinator is trained well it provided enough 'difficulty' in playing that it really does engage the performer. Once the training is over, and you really start to explore, it becomes a process of finding new sounds and spaces that the mapping has created and then trying to include those into your vocabulary of performing with the instrument."

Balancing Surprise and Complexity with Predictability and Control

The abilities to temper the degree to which the Wekinator behaved in surprising and complex ways, and to enforce more predictable and controllable behavior, were also important to composers. One wrote, "I think that the most satisfying mappings are the ones that establish a compromise between the ability to easily and reliably get back to a particular state and the ability to create new and unexpected outputs." Another wrote, "... There seems to be a happy medium where there is linear, logical control, and more intuitive, unpredictable control. For example when I play an acoustic string instrument, say a cello, I can predictably produce pitch by placing my finger at a certain point on the string. However this is still dependent on a number of other more subtle features (similar to things that have been incorporated into the blotar model) like bow pressure, placement of the bow on the string, the angle of the bow, the movement of the finger holding down the string. The interactions are not

necessarily 'linear' or obvious. I think the combination of the two is what makes an acoustic instrument exciting, and the same seems true here."

Composers used the Wekinator's interface capabilities for selecting which features influenced which parameters, and for influencing which training data influenced which parameters, to enforce greater control and predictability. One composer describes his strategy: "At first I tried to single out each output parameter individually and train the system one at a time, although ultimately I found that it proved to be most useful when I didn't try to mico-manage the inputs/outputs, and just gave it a snapshot of all the features and then let the network sort itself out. At this point I think I've reached a compromise between the two ways of working by using the 'Model Setup' panel to toggle on or off which features I want to contribute to each parameter individually. This way I can have some variables that react to the input features in a more predictable one-to-one relationship, and others that are a little more mysterious."

As discussed in Section 4.4.3, composers also used the training dataset to manage the complexity of the trained models' functions. Composers used the initial training dataset to delineate the boundaries within which they were willing to allow the neural networks to surprise them, and they added training data both to enforce particular, more predictable behaviors, and to make the models more complex.

An Invitation to Play

"Play" was a common word in composers' descriptions of how they interacted with the Wekinator and what they liked about it. "Play" and "playability" were important themes related to prioritizing gestural interaction, and these terms were used to positively characterize the tone of the interaction as collaborative, informal, and exploratory. According to one composer, "It feels like design software that lends itself to the imagining of 'playful' instrument to me. By 'playful' I mean both whimsical as well as embodied and outside the box." According to another, "The way that the Wekinator becomes like a toy and collaborator makes it very appealing."

Advantages Over Other Tools

Composers strongly agreed that the Wekinator offered expressive and practical advantages over other mapping techniques they had employed. Using a 5-point Likert scale, they rated their agreement with the statement "The wekinator allows me to create more expressive mappings than other mapping techniques" as 4.5, on average ($\sigma = 0.8$). They rated their agreement with the statement "The wekinator allows me to create mappings more easily than other mapping techniques" as 4.7, on average ($\sigma = 0.5$).

Asked about why they would use the Wekinator versus some other software to create mappings for a new composition or instrument, six composers mentioned that they would choose to use other software if they wanted very simple mappings, and they would choose the Wekinator if they wanted something more complex. For example, one composer wrote, "If the mapping were super simple, for example a foot switch

that simply steps though a series of presets or triggers something, then I would not use Wekinator, and probably just use a straight forward one-to-one mapping using Max." Other composers echoed these ideas, expressing that they had prior experience using Max/MSP or ChucK to create one-to-one mappings using very simple mapping functions (e.g., linear functions or threshold triggers). They did not view the Wekinator as necessary for designing such mappings accurately or efficiently, and some felt that the ability to explicitly design the function (by using code and not the Wekinator) granted them a higher, more desirable degree of control.

Other reasons composers cited for choosing the Wekinator echo the discussion above: they viewed the Wekinator as useful for accessing unpredictable behaviors, designing theatrical and musically sensitive performances, easily building controllers for synthesis algorithms with many complicated parameters, and privileging abstraction, physicality, and interaction during both instrument design and performance.

4.4.5 Improvements to the Wekinator

The above discussion of the Wekinator and the qualities that people liked about it pertain to the state of the software at the end of the ten-week participatory design process. Throughout the eight design revisions, many changes were made to the software in response to composers' suggestions and to the difficulties that arose. In this section, we detail the most significant improvements that were made to the Wekinator, grouped by the motivations for their implementation. We finish this section with a discussion of the proposed improvements that we still plan to implement in the future.

Supporting Greater Control and Constraint over the Supervised Learning Problem

At the beginning of the design process, the Wekinator offered no mechanisms for limiting the number of parameters affected by an input feature (i.e., feature selection) or for limiting the number of parameters affected by a training example. All features and all training examples were used in the construction of the training dataset for all models. In the second meeting, a composer mentioned that she was interested in controlling different compositional processes using different types of inputs, for example controlling one process with a gestural controller and one with a singer's voice. Another composer indicated that he was interested in using different gestural properties to control different synthesis parameters, even though all synthesis parameters might be controlled using the same input device and the same input features from that device. For example, in using the laptop motion sensor to control the pan (between the left and right audio channels), pitch, and randomness parameters of a Max/MSP patch, he knew he wanted pan to be controlled by tilting the laptop left and right. Even though he also planned that tilting left and right would affect the other sound parameters, he wanted to be able to create training examples for panning control without simultaneously thinking about how the other sound parameters would be affected by those gestures.

Therefore, both feature selection and training data selection functionality (described in Sections 3.4.4 and 3.3.5) were added to the Wekinator to allow composers greater flexibility in applying the Wekinator to the types of musical control applications that they envisioned. We also added the ability for the user to initiate training of only a subset of models at any given time, so that users who were focusing on the refinement of a single model or subset of models did not have to wait through the training of models that were irrelevant to them at that moment.

The architecture of earlier versions of the Wekinator had left open the possibility of using a single neural network with multiple outputs to control multiple continuous parameters. However, the possibility that different parameters would be affected by different subsets of features and/or training data, and composers' desire to be able to approach the training and refinement of different parameter models in separate stages, required the use of independent neural networks for different parameters.

We also redesigned the system backend and GUI so that users could specify that different learning algorithms would be used for different discrete parameters, for example a support vector machine for one discrete parameter and a decision tree for another. Originally, just one algorithm would be selected for all discrete parameters, but this constraint became especially problematic when different parameters were trained using very different numbers of features and training examples.

Taking Advantage of Discovery

Composers often relied on the neural networks to provide a way of gesturally exploring the high-dimensional parameter space of synthesis algorithms like the blotar. They enjoyed discovering new sounds that they hadn't previously imagined, and that would have been hard to discover by setting parameter values manually through a GUI. By adding the parameter clipboard to the Wekinator (Section 3.4.6 on page 74), and adding the "add to clipboard" button to the "Run" subview of the "Use it!" interface (Figure 3.29 on page 73), we provided users with the means to easily capture and save parameter settings that they liked. These parameters could be used in a later composition in which the sound was appropriate, or they could be fed back into the current Wekinator training set to make them more prominent or more easily attainable (e.g., by matching them with several variations of an input gesture).

Using the parameter clipboard as a playalong score also allowed composers to take advantage of new sonic gestures, or changes in sound parameters over time, that they discovered during experimentation. Prior to the participatory design process, the only way to perform playalong example recording was through the use of a ChucK score. Changing the contents of the playalong score involved editing the ChucK code outside of the Wekinator environment, then stopping and restarting the ChucK component of the Wekinator. This process could take a considerable amount of time and did not allow composers to easily test the playalong score by listening to it as they wrote it. After the parameter clipboard was added and composers could construct, test, and modify their playalong scores on-the-fly, within the Wekinator GUI, nobody used the ChucK playalong scores anymore, even though they in principle offered a greater degree of control over the construction of the score. This suggests that, for

these composers, the additional control and flexibility offered by the ChucK scores did not compensate for the disruption and overhead incurred by switching out of the Wekinator GUI (and the modes of interaction supported by it) and into coding in ChucK.

Supporting Abstraction and Physicality

Another motivation for implementing the parameter clipboard playalong score was to better support abstraction and physicality. When constructing a playalong score in ChucK, composers had to explicitly consider the numeric parameter values and then write code to specify how they changed over time. In contrast, a user can add to the parameter clipboard any set of parameters whose sound he likes by clicking on the "add to parameter clipboard" button of the "Collect Data" interface, shown in Figure 3.23 on page 67; the user can find sets of parameters whose sound he likes by either editing their values in the Wekinator GUI and listening to the results, or by manipulating a control GUI for an external synthesis environment such as Max/MSP. (As discussed in Section 3.3.4 on page 54, updates to parameter values initiated in another environment can be automatically and immediately sent to the Wekinator, where they are reflected in the Wekinator GUI and able to be added to the parameter clipboard.) As mentioned above, playing the parameter clipboard as a playalong score also allowed composers to practice gesturing along with the sounds before recording training examples. Either the gestures or the sound parameters could be easily changed until the composer liked both the sound and the "feel" of the score and gestures together.

Currently, when the parameter clipboard is used as a playalong score, each parameter set is played for a set amount of time, then the parameters are immediately changed to their next values. We are currently implementing functionality that will allow users to specify that parameters should smoothly change from one setting to another over a period of time, allowing training examples created during playalong to capture more detailed information about how changes in gesture should correspond to changes in parameters.

The "meta-features" for computing first- and second-order differences (Section 3.3.5 on page 56) were also added in response to composers desiring features that captured what they perceived to be musically relevant aspects of their physical gestures (i.e., velocity and acceleration). Musicians explicitly manipulate the velocity or acceleration of their motions in certain musical techniques; for example, the sound produced by the bowing of a violin is dependent on both the position of the bow against the string and the velocity of the bow. Most gestural input devices, however, typically transmit only information about their current state, and the learning algorithms in the Wekinator do not consider how features change over time. Therefore, without meta-features, a Wekinator user who wished to use the velocity or acceleration of a gesture as a control mechanism would have to write a custom ChucK or OSC feature extractor that computed these measures on the input features extracted from the controller. As a result, the meta-features were added to allow performers to easily employ estimates of velocity and acceleration for any input device. The

smoothing and history buffer meta-features were added to provide access to other temporal features that may be useful to composers.

Improving Usability for Diverse Users

Composers in the group had a wide variety of comfort levels in using the computer for certain tasks, and a diversity of ways of approaching and thinking about their work. Furthermore, the majority had no prior exposure to machine learning. Therefore, a significant amount of meeting and development time was devoted to understanding how to make the Wekinator more usable and more understandable by a wider variety of users.

Initially, the Wekinator did not include any interface support for configuring the ChucK component of the system or for running it from the Java GUI. Instead, users encoded a ChucK configuration in a relatively succinct ChucK file (about ten lines of code). This file indicated which ChucK implementation of each Wekinator component (feature extractors, synthesis algorithms, playalong scores) would be loaded. Then, the user launched the ChucK component of the Wekinator manually from the command line, in an independent step from launching the Java portion of the Wekinator. This mechanism for running ChucK separately from Java had several advantages: users could start and stop the ChucK and Java components of the Wekinator independently, and they could view error messages from the two components separately in different terminal windows. The back-end implementation of the ChucK and Java components was also simple, in that they did not have to keep track of each other's state.

However, some composers in the group found it hard to get started using the Wekinator, as they found writing ChucK and/or using the command line to be a high barrier to both starting the system and reconfiguring the system when they realized they wanted to change something. A substantial amount of development was therefore devoted to providing support within the Wekinator's Java GUI for configuring, running, and stopping ChucK. (The final interface for this functionality was described in Section 3.4.2 on page 59.) The console window, discussed in Section 3.4.2 on page 60, was also added to display both Java and ChucK output to the user, so that users were still able to debug the ChucK code launched by the Wekinator.

One composer liked being able to specify the ChucK configurations as code files, and he liked having the ability to stop and restart ChucK in the command line. Therefore, the redesign of the Wekinator maintained these abilities, and exposed an alternative user interface to allow manual setup of the OSC connection between the Java component of the system and the user-launched ChucK component, within an "Advanced" tab of the "ChucK Configuration" GUI tab shown in Figure 4.4.

Apart from this one composer, though, the rest of the composers who were initially comfortable using ChucK configuration files and command line launching switched to using the Java GUI for ChucK configuration and launching. Even though this feature of the system was designed to meet the needs of less technical users, once it was implemented, it provided an interface that many of the more technical users found to be more convenient or more usable.

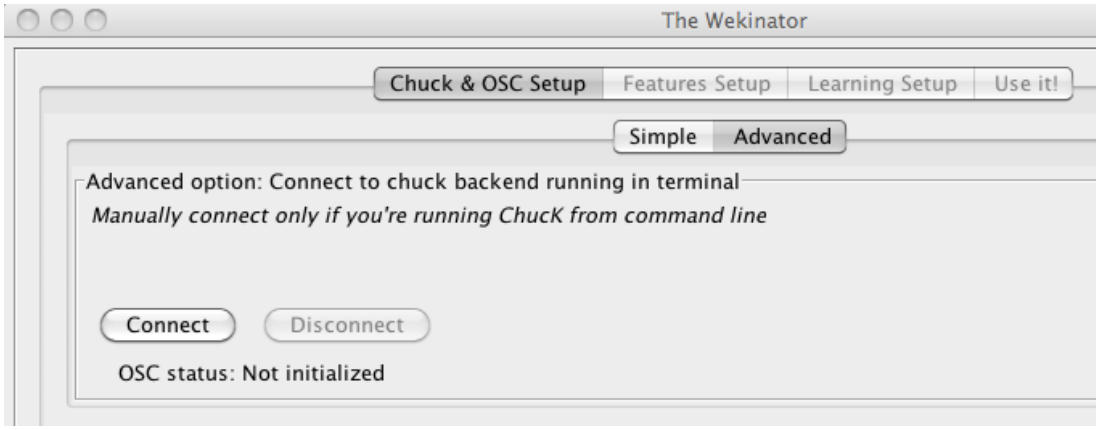


Figure 4.4: The "Advanced" view of the "Chuck & OSC Setup" tab is used when the ChucK component of the Wekinator is launched separately from the command line. The Java GUI is only used to initiate OSC communication with the ChucK component.

Another set of design revisions focused on hiding aspects of the system state or configuration options from users who found them irrelevant or overwhelming. For example, the initial interface for feature selection consisted of an on-screen matrix of checkboxes, where each row corresponded to a feature, each column corresponded to a parameter, and each matrix element indicated whether that row's feature was selected for that column's parameter. The matrix offered a single interface for users to specify their intended feature selection for all parameters, and it provided a compact representation that visually indicated the entire state of feature selection. However, several composers found the matrix to be overwhelming; they had a hard time understanding how to use it, and they felt pressure to fill out the many checkboxes "correctly" before using the Wekinator to start building the models. In response to this problem, we created a separate user interface for feature selection for each parameter, and this interface was hidden by default. (The feature selection interface is now accessed by clicking the "View & choose features" button for each parameter in the "Learning Setup" tab, shown in Figure 3.21 on page 65). Also, the console window was made to be not visible by default, as some users found the output intimidating and were not certain when the output signaled errors that they needed to fix, and when it could be ignored. Logging behavior was also added to the Wekinator to address this issue; error messages and status messages were printed to a text file that was never visible to the user, but which the user could email or show to us when he wanted help tracing the cause of an error he encountered.

A significant portion of the group's questions and discussion centered around the language used in the Wekinator interface and documentation. Several people mixed up the meaning of "features" and "parameters." While "features" is a fairly common term in machine learning, it is not used in computer music, and users sometimes thought of the values they sent to the Wekinator from the feature extractors as "parameters" controlling the Wekinator. However, the group did not come to a

consensus on terms that might be better, so we kept this terminology and added more documentation to the Wekinator's instructions to clearly define these terms and graphically illustrate how they are used in the software. Some composers also did not find the distinction between "discrete" parameters (i.e., those controlled by classifiers) and "continuous" parameters (i.e., those controlled by neural networks) to be clear. "Integer-valued" and "real-valued" were proposed as alternative category names, but no consensus was reached on whether these would be better.

At least one composer had difficulty reconciling her own understanding of building an instrument or composition with conventional supervised learning terminology and the workflow for training a model from a dataset: "I think making the aspects of Wekinator training that encompass spreadsheets and data sets a bit more organic or transparent to those of us who don't approach creating art work in this manner. I just don't ever think 'spread sheet' or 'data set' when I'm creating work and this may be because my work is usually simultaneously conceptual and narrative." We chose not to change the Wekinator to obfuscate the underlying supervised learning process, both because we wanted the Wekinator to be clearly understandable by people who were familiar with Weka and with supervised learning, and because we failed to come up with an alternative metaphor for explaining system behavior that was any clearer than discussing the supervised learning process itself. However, we are involving this composer in the ongoing revision of the Wekinator instructions and documentation to make it as clear and accessible as possible to people who might approach the Wekinator with similar perspectives.

Another avenue of work undertaken in an attempt to make the Wekinator more accessible to composers was the implementation of many example ChucK and OSC feature extractors, synthesis algorithms, and other controllable processes. These modules allowed users to experiment with the Wekinator for a wider variety of applications without having to write any code themselves, and it also provided clear examples to help users get started writing their own code for use in the Wekinator. While many of the example modules were written by us, some were developed in response to particular requests by composers, and others were developed by composers themselves and shared with the group via the email list.

Improving Compatibility with Composers' Preferred Tools

One composer in the group worked almost exclusively in Max/MSP, and he had previously developed many Max/MSP software tools for use in his own compositions. Many other composers worked frequently in Max/MSP, and people expressed interest in using Max/MSP for both synthesis and feature extraction. For example, composers were interested in using the Wekinator to control compositions packaged as Max/MSP "patches" as well as synthesis algorithms packaged as Max/MSP "externals," including the blotar; they were also interested in taking advantage of prior work integrating novel gestural controllers into Max/MSP. The SpaceNavigator, in particular, does not operate as a HID device, but we were able to use it as an input device by using existing Max/MSP code to extract features describing its state, then sending these features from Max/MSP to the Wekinator via OSC.

Therefore, another significant component of the design process focused on making it easier for composers to use the Wekinator in conjunction with Max/MSP. Features added at the request of composers included OSC communication of synthesis parameter values from Max/MSP back to the Wekinator (described in Section 3.3.4 on page 54), OSC communication of feature names from Max/MSP feature extractors to the Wekinator (for display in the Wekinator's interfaces for feature selection and the Feature Viewer), OSC commands for controlling training and recording (described in Section 3.5.2 on page 78), and the user interface for specifying all OSC parameter names and types (shown in Figure 3.17b on page 61).

Although these added software features were used in the context of communication between the Wekinator and synthesis patches or feature extractors running Max/MSP, users employing a feature extractor or Wekinator-controlled process in any other OSC-capable environment are equally able to take advantage of these features; they do not rely on the Max/MSP environment in any way, only the ability to send and receive OSC messages.

Providing More Information About the System State

Composers requested the addition of several interface components to improve their ability to understand the current system state, for identifying errors as well as gaining information to help them use the Wekinator more effectively. We added the Feature Viewer (described in Section 3.4.3 on page 63) to aid in the detection and diagnosis of errors in the feature extractors, in OSC communication, or in the Wekinator itself. The Feature Viewer also allowed people to learn more about the relationships between their gestures and the feature values, for example whether a single motion sensor axis might be used to sense a particular laptop tilt gesture. We added progress bars for training and evaluation (pictured in Figure 3.27 on page 71 and in Figure 3.30 on page 74) so that users could obtain an estimate of the total computation time remaining and use that to determine whether to cancel the computation or wait for it to complete. The spreadsheet-style data editor (shown in Figure 3.25 on page 69) was initially developed as a way for people to view the dataset, for example to check which parameter values they had already used in training, and to check that the feature extractors were working during the training example recording process. We added metadata fields to the training data, indicating the time at which each training instance was recorded and the "recording round" of each instance, to provide additional context for understanding the dataset.

The graphical data editor (shown in Figure 3.26 on page 70) was not completed during the participatory design process, but its development was motivated in part by composers' desire for an interface for understanding the training data that provided a more visually informative alternative to the spreadsheet viewer.

Reducing Barriers to Rapid Prototyping and Exploration

Several improvements did not add new functionality to the Wekinator, but they made the process of model building and editing faster and easier. The original version

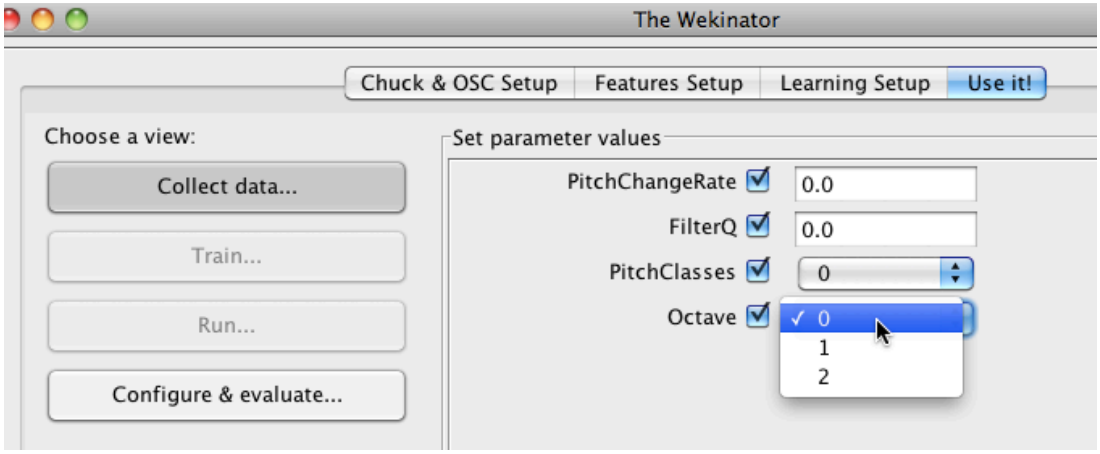


Figure 4.5: Discrete parameter values are set using a drop-down list populated only with legal values.

of the Wekinator did not display names for each parameter in the "Collect Data" pane pictured in Figure 4.5, and it allowed the user to enter any numeric value for any parameter, leading to problems when users mistakenly set non-integer or out-of-range values for discrete parameters. Therefore, functionality was added to retrieve parameter names, types, and (for discrete parameters) ranges from both ChucK and OSC processes, display the parameter names in the Wekinator GUI, and restrict the values able to be entered for discrete parameters through the use of a pre-populated drop-down list (Figure 4.5).

When composers had difficulty efficiently training gestural controllers that used both hands, we added support for the foot-pedal control discussed in Section 3.4.5 on page 68.

Because composers commonly used modification of the training data and retraining as a way to modify the mappings, adding the "training round number" metadata field to the training dataset allowed people to easily undo changes they didn't like, by deleting all examples of a training round within the spreadsheet viewer.

As they worked, some composers frequently saved the trained models that they liked so that they could build many candidate models and pick their favorite. We added infrastructure to make dealing with saved files more convenient, including default file extensions for all Wekinator-created files, a directory infrastructure for organizing files (e.g., files for trained models were saved to a different directory than saved feature configurations), and more graceful error reporting when a user tried to load a file that was incompatible with the current state of the Wekinator (e.g., a learning system whose number of features does not match the number of features being extracted).

Functionality was added to enable the Wekinator to maintain aspects of its system state in between uses of the program. This included information about the last-used ChucK configuration, so that the last ChucK configuration is reloaded each time the Wekinator is run, and about the location where each Wekinator file type was last

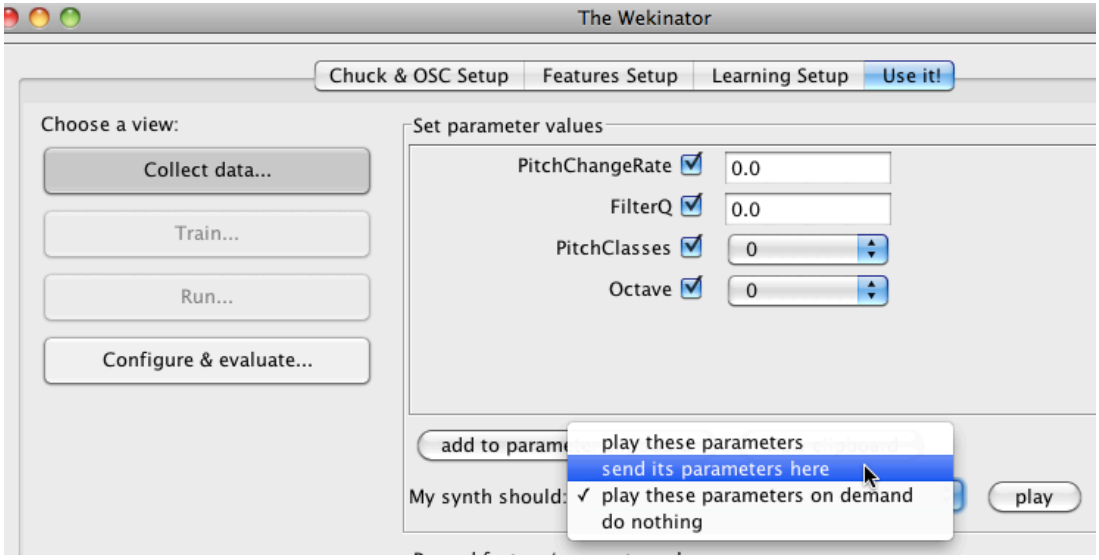


Figure 4.6: The drop-down list allowing the user to specify whether and when parameter values set in this GUI will be sent as control values to the ChucK synthesis class or OSC process.

saved, so that a user attempting to load a Wekinator component from a saved file is presented with a file chooser that defaults to the last-saved location.

To efficiently accommodate composers' different approaches to experimenting with the synthesizer sounds during use with the Wekinator, we added the drop-down list shown in Figure 4.6. Some composers working with Max/MSP patches preferred to set the synthesis parameters directly within the Max/MSP environment, for example using sliders and number boxes attached to the synthesis object in their Max patch. These users did not necessarily want changes in the parameter values in the Wekinator GUI to result in new parameter values being set in Max. On the other hand, some composers preferred to experiment with different parameter settings by changing the values in the Wekinator GUI; for some synthesis algorithms, they were interested in auditioning sets of parameters together, and for other algorithms, they were interested in hearing changes caused by updates to each parameter individually. These different approaches were accommodated by the drop-down options "send its parameters here," "play these parameters" (i.e., as soon as the values are updated), and "play these parameters on demand" (i.e. only when the "play" button is hit).

4.4.6 Future Improvements Proposed

Development of the Wekinator during the participatory design process was focused on changes that most improved the usability of the system for the greatest number of composers within the given time constraints. Composers also proposed and participated in the refinement of many good ideas for further improving the software, beyond what was accomplished during the study.

One major area for future improvement involves making the Wekinator more suitable as a general-purpose tool for audio classification and music information retrieval. The current ability of the Wekinator to be applied to generic audio analysis problems is most limited by the relatively small number of audio features built into ChucK, and by the inability to apply standard feature extraction packages used in music information retrieval to feature extraction from real-time audio. During the participatory design process, we worked with composers interested in audio analysis to build a ChucK vowel formant feature extractor, and we created a Wekinator-compatible Max/MSP patch that extracts additional audio features using the analyzer~ object (Jehan and Schoner 2001). Both of these audio feature extractors are distributed with the Wekinator to help users get started in applying the Wekinator to working with audio. However, the use of the Wekinator as an audio analysis tool will still be greatly improved by future work either adding additional audio features to ChucK or implementing a real-time feature extraction suite in another environment.

Composers' early experimentation with the Wekinator also revealed the current infrastructure to be problematic for classification problems involving fine-grained, time-sensitive class labeling. For example, one composer spent some time attempting to teach the Wekinator the difference between different types of drum strikes, using an audio signal from a piezo pickup on the drum membrane. While the features and classification algorithms seemed to be suitable for this type of problem, based on prior research in drum classification (e.g., Tindale 2004), the Wekinator's approach to audio feature extraction and interfaces for training data creation were not sufficient. Specifically, this problem required either a way to extract feature vectors for training and classification only immediately following drum hits, and/or a mechanism for annotating all extracted features with the class labels (including a label for "no hit" in between drum hits), at a fine-grained level. The graphical editor that was implemented later, which is shown in Figure 3.26 on page 70, provides a partial solution, in that users can easily edit and add class labels to the recorded training instances. However, this is only practical when the graphical editor's display of the feature values provides sufficient visual cues to the human user regarding where the class labels should fall. In the future, the Wekinator might also record the raw audio signal (or even a webcam video of the user) that is temporally aligned to the recorded training example feature vectors, so that the human has all the relevant information needed to perform precise class label annotations.

We also discussed with composers the possibility of adding a two-dimensional visual projection of the training data, displaying information about instances' relative distances in feature space as well as their class values or labels. Baker et al. (2009) implemented this type of visualization in a system for end-user training of a document labeling model, in order to provide users with useful information regarding the extent to which differently-labeled instances overlapped in the feature space. Such overlapping can indicate to a user that his training examples are noisy or mislabeled, or that his features are not adequate for representing the classification problem. Composers seemed interested in using such a tool to understand their data better and identify when their training examples or features might be to blame for a trained classifiers' poor performance.

Additional features could be added to further reduce barriers to rapid experimentation in the Wekinator. For example, improvements could be made to allow users to more easily change the number of parameters on-the-fly, or reload saved models from learning systems with different numbers of parameters and features.

Some composers desired additional support for controlling the Wekinator from alternative interfaces, including MIDI controllers, user-designed GUIs, and Max/MSP patches. The implementation of such control might involve, for example, the use of OSC communication from the Wekinator about its state, and OSC commands for configuring feature extraction and loading learning systems. We believe that such functionality will make the Wekinator more flexible for a variety of applications, including research applications, and we intend to add this in the future.

4.5 Discussion

4.5.1 The Wekinator as Compositional Tool

At the culmination of the participatory design process, most composers were able to very effectively employ the Wekinator in creating new instruments and interfaces for the continuous gestural control of sound synthesis algorithms. Composers highly agreed that the Wekinator enabled them to create gesture-to-sound mappings more easily than other techniques, and to create mappings that were more expressive than other techniques. In particular, composers valued the speed and ease with which they could create mappings, the ways that the Wekinator supported embodiment and abstraction, and access to a balance of surprise, complexity, predictability, and control. Two of the composers went on to use the Wekinator in publicly performed compositions, which are discussed in Chapter 7.

The usability and usefulness of the Wekinator were supported by the Wekinator's user interface, the ability to interactively apply supervised learning, and the underlying multilayer perceptron algorithms. For example, the use of neural networks allowed composers to easily build complex, nonlinear, and surprising mappings from the training data. The ability for users to iteratively record and edit training data, train the models, and "play" the resulting models, all within a single user interface, enabled easy and fast experimentation with different mappings. The ability for users to create training data by demonstrating gestures and by gesturing along to playalong scores supported an embodied approach to designing and creating mappings. Other interface features, including those for informing users of the system state and integrating with other compositional tools via OSC, allowed users to build and modify mappings more quickly, and to identify errors more easily.

Two composers had notably less success than the others in applying the Wekinator to their own projects. One of these composers experienced ongoing difficulties reconciling the supervised learning workflow and terminology with her own approach to composition, which she described as more narrative-driven and organic than what the Wekinator seemed to suggest and accommodate. The other composer was more interested in using audio input than gesture, which was problematic for the reasons

described in the previous section, and spent much less time experimenting with the Wekinator overall. These two composers were still able to make valuable contributions to the design process, by providing feedback about what they found challenging, and about the types of projects they wanted to be able to do with the Wekinator in the future. Some of their concerns and difficulties have been addressed by changes to the software and documentation, and we hope to continue to involve them in the project to further improve the Wekinator's usability.

The participatory design process resulted in many ideas for improvements to the Wekinator software, some of which were implemented and contributed greatly to its usefulness and usability, and some of which we still plan to implement in the future. However, there still remain questions regarding how to improve the usability of the Wekinator that were unanswered during this work, and which we intend to address in future research. Notably, there exist many challenges regarding how to make systems for interacting with supervised learning algorithms most useful and understandable by users who lack a background in machine learning. The biggest challenges faced by composers related to the use of machine learning or computer science terminology that had no counterpart in music (e.g., "features," "discrete" and "continuous"), not knowing how to take action when problems occurred (e.g., using fewer features or training instances when neural network training took an unacceptably long time, or changing a classification algorithm or its parameters when a classifier performed poorly), and not understanding the types of learning problems that were easy or hard to model. Some of these difficulties (for example, not understanding the types of functions learnable by neural networks) disappeared as users gained more experience with the system; others (for example, not understanding how to keep neural networks from taking a long time to train, or not understanding when one classification algorithm might be better than another) did not. Future research focusing on making supervised learning systems more accessible, through changes to interfaces, documentation, or even the algorithms themselves, seems important not only to users of the Wekinator but to end-user machine learning systems in many domains. We discuss some ideas for this future research in Chapter 8.

4.5.2 Generative and Explicit Mappings

In light of existing work categorizing mapping strategies as generative or explicit, many of our observations regarding composers' interaction with the Wekinator appear at first glance to be rooted in the Wekinator's nature as a generative mapping creation system. Generating mappings from training examples inherently supports an embodied and enactive (Wessel 2006) approach to the mapping specification processes, in that gestures can be used to create the training examples. The specification of mapping via example also enables many low-level details to be abstracted away, speeding up the mapping exploration process, facilitating use by non-programmers, and freeing the composer to focus on the creative process. The use of neural networks, which are capable of introducing great nonlinearities into the learned functions, also contributes to serendipitous discovery and efficient production of complex mappings.

Overall, we conclude that many qualities of the Wekinator that are most helpful in compositional interaction are well-supported by properties of the generative mapping strategy. However, a user who desires a more specific mathematical relationship between gestural features and sound parameters may be frustrated by the difficulty and inefficiency of implicitly encoding this mapping via training examples; it is therefore also obvious that the suitability of example-driven mapping generation is contingent on users' compositional goals.

On closer examination, one can imagine interfaces that do not enforce a clear dichotomy between generative, example-driven and explicit, function definition-driven mapping creation, supported by underlying algorithms that are difficult to mathematically categorize as purely generative or explicit. For example, simple linear or polynomial regression algorithms could be embedded in an interface that allows the user to dynamically switch between supplying examples and explicitly editing equations.

4.5.3 Influence of Technology on the Composer

Nearly all composers characterized their interaction with the Wekinator as more complex than an exercising of control over technology; the ways the software challenged and influenced them were important aspects of their experience as users. Composers often relied on the Wekinator as a tool that could quickly take a "sketch" of the learning problem, encoded by a few training examples, and turn it into a fully-functional mapping. Rather than relying on the neural networks to fill in this sketch according to their preconceived plans, composers relied on the neural networks to provide interesting, complicated, and unexpected mapping behaviors that incorporated complexities and difficulties similar to those encountered in the playing of an acoustic instrument, and that provided access to unimagined and inspiring sounds and gesture-sound relationships.

Through using the system, composers also learned about what interaction strategies produced mappings that they liked (for example, "At first I tried to single out each output parameter individually and train the system one at a time, although ultimately I found that it proved to be most useful when I didn't try to micro-manage the inputs/outputs, and just gave it a snapshot of all the features and then let the network sort itself out."). They also learned that certain mapping strategies (such as enforcing a linear relationship between features and parameters) were hard to satisfactorily produce using the neural networks. As a result, their goals and subsequent interaction with the system often changed course according to how the trained models behaved and according to their improving knowledge about the effort needed accomplish different tasks.

4.5.4 Further Discussion

In Chapters 8 and 9, we further discuss the implications of this work regarding the larger context of interaction in machine learning and machine learning in creative contexts.

4.6 Conclusions

In this chapter, we discussed our experiences working with a group of composers as they experimented with the Wekinator and helped drive improvements to the software in a participatory design process. In this process, composers were engaged in learning to use the software, applying it to their work, and communicating to us about their experience, feedback, and ideas. We were able to make many changes to the software to make it more usable and useful to the composers. Through discussion with and observation of the composers, we also learned a great deal about how composers relied on and were influenced by technology in the process of composing new instruments, and we learned about how the interactive supervised learning process enabled by the Wekinator was useful in supporting their compositional goals and needs.

Chapter 5

Teaching Interactive Systems-Building with the Wekinator

5.1 Introduction

In this chapter, we discuss our study of the Wekinator as a tool for teaching interactive systems-building to undergraduate students enrolled in the Princeton Laptop Orchestra course. We focus on students' use of the Wekinator in completing a structured systems-building assignment, as well as a longitudinal analysis of students' work with the Wekinator over the course of the semester. Through logs of students' interactions with the software and their written feedback, as well as the final outcomes of their coursework, we examine how they used interactive learning to create musically expressive and accurate models, the ways that the Wekinator influenced their work, and the efficacy of the Wekinator as a teaching tool.

We begin this chapter by providing background on the Princeton Laptop Orchestra course. We discuss how students' work with the Wekinator fit into the goals of the course, and we present our research goals for our study of its use in this context. We describe the method used for studying students during their use of the Wekinator in the midterm assignment, and we present our observations regarding the interactive systems the students built, the actions they performed, how they used their time during the assignment, their strategies for model building, the ways that they learned and adapted, their degree of success in the assignment, and the effects of task type and order on students' behaviors. Drawing on our observations and other experiences in the course, we discuss the usefulness of the Wekinator as a teaching tool, the relevance of task type in studying and supporting interactive supervised learning, and improvements to the Wekinator suggested by this work. We later draw on this work in Chapters 8 and 9, in discussing algorithm and interface design in interactive supervised learning, the use of interactive machine learning by novices and in creative work, and the larger role of human interaction in applied supervised learning.

Figure 5.1: One of the first PLOrk performances in 2006.

5.2 Background, Motivation, and Goals

5.2.1 The Princeton Laptop Orchestra

The Princeton Laptop Orchestra (PLOrk), pictured in Figure 5.1, is an undergraduate teaching initiative and performance ensemble, created in 2005 by Princeton faculty members Dan Trueman and Perry Cook (Trueman et al. 2006). Drawing inspiration from historic electronic performance ensembles such as the League of Automatic Composers and the Hub (Bischoff et al. 1978; Brown and Bischoff 2002), as well as from Trueman's own work creating digital meta-instruments such as the BoSSA (Trueman and Cook 2000), a motivation for forming PLOrk was to experiment with making music with a larger group of performers playing laptop-based meta-instruments (Trueman 2007). Another inspiration for PLOrk was work by Cook and Trueman (1998), Wessel (1991), and Trueman et al. (2000), exploring how to design new electronic instruments with a physical presence more like that of acoustic instruments, specifically through the use of spherical speakers. In PLOrk, many human laptop performers share the stage, each using their own laptop and hemispherical speaker, shown in Figure 5.2.

At its inception, the PLOrk ensemble consisted of fifteen human laptop players. Through the years since 2006, the size of the ensemble has grown to over 25 performers, and the ensemble has performed both full-ensemble and "chamber" works (for fewer performers) composed by Princeton faculty, students, and alumni, as well as guest composers. Notable past collaborators include renowned composers Paul Lan-

Figure 5.2: Each PLOrk student uses his or her own laptop and hemispherical speaker.

sky and Pauline Oliveros, tabla virtuoso Zakir Hussain, and experimental electronica duo Matmos.

In addition to being an active performing ensemble, the Princeton Laptop Orchestra is an undergraduate educational initiative. PLOrk was first taught as a course at Princeton University in Autumn 2005 as a Freshman Seminar. It has been taught every year since then as an undergraduate course in Computer Science and Music, open to students of any major and any academic year.

Since 2008, the PLOrk class has included assignments and weekly lectures devoted to exposing students to a variety of theoretical and practical topics in computer music composition and performance.[1] The class introduces student to topics including object-oriented design and programming (primarily using the ChucK language; see Wang and Cook 2003), software engineering, signal processing, audio synthesis, and interactive systems-building. A goal of the course is to give students from non-technical majors exposure to these areas of computer science and engineering, and to give students in technical majors practice in applying their expertise to new domains and in creative ways.

[1]Syllabi for the last three offerings of the course may be found online at `http://www.cs.princeton.edu/~fiebrink/thesis/resources.html`.

5.2.2 Supporting and Studying Interactive Systems-Building in PLOrk

In Spring 2010, the PLOrk course emphasized interactive systems-building. In many assignments, as well as in the midterm and final projects, students built new musical interfaces that could be used in performance and as interactive installations. These interfaces ranged from very simple, for example using the laptop's internal accelerometers to control the pitch of a synthesis algorithm in an assignment, to quite elaborate, for example using a custom vision-based motion tracking program to sonify the walking patterns of visitors to the music building in a final project installation. Interactive systems-building projects were used both as a tool to help students learn course concepts (e.g., software engineering and signal processing), and as an end in themselves, to empower students to create interactions they found to be fun, compelling, and expressive.

One of the topics emphasized in the course was the "mapping problem," described in Section 2.2.3. Students were first taught about "explicit mappings," in which gestural controllers are built by explicitly programming the ways that gestural controller or sensor values drive the parameters of a synthesis algorithm. Students built several explicit mappings in course assignments by using SMELT repository code (Fiebrink et al. 2007) to extract gestural input signals from USB HID devices (i.e. standard input and gaming devices; see Section 3.3.2) and native laptop inputs, then writing their own ChucK code to change sound synthesis parameters based on the values of these inputs.

Students were also taught about "generative" mappings that construct the gesture-to-sound mapping function from a set of training examples, and they were given a high-level introduction to how machine learning algorithms can be used to generate mappings from training data. Prior to the Wekinator, there existed no appropriate tools to allow students (especially those with limited programming ability and machine learning knowledge) to create generative mappings for using arbitrary input devices (including HID devices, audio and video inputs, and other native laptop inputs) to control arbitrary synthesis programs. The Wekinator is therefore an attractive tool for enabling students to quickly and easily build working interactive systems and to learn about generative mappings and machine learning in a hands-on way.

The goal of the work described here was to study the first use of the Wekinator as a tool for teaching interactive systems-building in an undergraduate classroom. The Wekinator was used in the PLOrk course to teach students about the breadth of ways that computers can be used in interactive performance, to enable students to build interactive systems that they enjoyed and found musically useful, to teach them about the tradeoffs between explicit and generative mapping strategies, and to foster the development of their object-oriented design, signal processing, and music programming skills as they designed feature extractors and sound synthesis programs to be used in conjunction with the Wekinator. In our study of students in the course, we hoped to discover how the Wekinator was useful in meeting these goals, and to discover what about it could be improved for future pedagogical applications.

Additionally, we aimed to learn about how students used the Wekinator in building interactive instruments and installations, including the actions they performed, the strategy they employed in building models, and how the Wekinator both supported and influenced their creative work.

5.3 Method

While we used the Wekinator throughout the latter half of the Spring 2010 PLOrk course, we focus our study here on our observations of and feedback from the 22 students completing the midterm assignment, which was their first and most structured assignment using the Wekinator. The midterm assignment served as a hands-on introduction to the Wekinator, and it prepared students to incorporate generative mappings into their compositions in the upcoming midterm performance. Prior to the assignment, students had five weeks of instruction in the ChucK programming language, and they had completed assignments in which they designed explicit mappings using gestures to control sound synthesis algorithms. In course lectures leading up to the assignment, students were introduced to generative mappings and to the standard machine learning algorithms employed by the Wekinator, and they saw demonstrations of the Wekinator performed by the author and by other instructors.

In the midterm assignment, students built two gestural musical controllers using the Wekinator: a continuous controller using neural networks, and a discrete controller using a classifier. Before beginning the assignment itself, the students followed a step-by-step tutorial walking them through how to build two simple instruments for controlling a synthesizer's pitch using the laptop's tilt. One of these instruments used a classifier for discrete pitch control, and the other used a neural network for continuous pitch control. Students worked individually on the assignment, and they had twelve days to complete it after it was assigned. As the goal of the assignment was to familiarize students with the Wekinator, students were informed that they would be graded on completing the assignment in a through and thoughtful manner, not based on the musicality of their models or the "correct" use of the Wekinator. They were encouraged to contact the course instructors for help and questions with the assignment.

The midterm assignment was broken into two parts, Part A and Part B, and the order in which students completed these parts was balanced across students. In Part A, students were asked to use the Wekinator's multilayer perceptron neural networks to build a new, continuous musical control interface that they thought was musically expressive. Students could pick among three pre-built synthesis algorithms, each of which had between three and nine parameters that affected the sound in non-linear and interdependent ways: a physical model of a bowed string instrument (Smith 1986; Cook and Scavone 1999), a physical model of a hybrid flute/electric guitar called the "blotar" (Stiefel et al. 2004), or an FM synthesis algorithm (Chowning 1973). Students could also pick among four gestural control methods for which the Wekinator provided built-in feature extractors: a GameTrak USB tether HID device (see Section 3.3.2 and Figure 4.1), a Logitech joystick HID device, the laptop's internal

accelerometers, or the webcam color tracking feature extractor described in Section 3.3.2.

After students had completed the construction of the musically expressive controller, they were asked to discuss their work in a series of five short-answer questions. Specifically, they were asked to explain their choice of gestural controller and synthesis algorithm, discuss their goals for how the instrument would be used to control sound through gesture, and describe their strategy for building an expressive model. Additionally, they were asked to rate on a 5-point Likert scale whether they were successful in building an expressive model, building a model whose gesture-to-sound mapping could be controlled in a predictable manner, and getting the Wekinator to learn what they wanted it to. Students were also invited to share any other comments on what they learned, what they found confusing, or anything else.

In Part B of the assignment, students were asked to use a classification algorithm to build a model performed reliable gesture classifications into any gesture categories of their choosing. The class labels output by model were used to control a simple, pre-built synthesis algorithm driven by a single discrete parameter. Students could choose either a melodic synthesizer whose parameter controlled pitch, or a drum machine whose parameter controlled the number of drum loops or layers to play simultaneously. Students could also choose among the same four gestural controllers as Part A: the tether, joystick, laptop accelerometers, or webcam color tracker.

After Part B, students completed five short-answer questions about their work building the reliable gesture classifier. These questions were identical to those answered after Part A, with the exception that they were asked about their success in building a model that provided reliable gesture classifications, rather than building a model that was musically expressive.

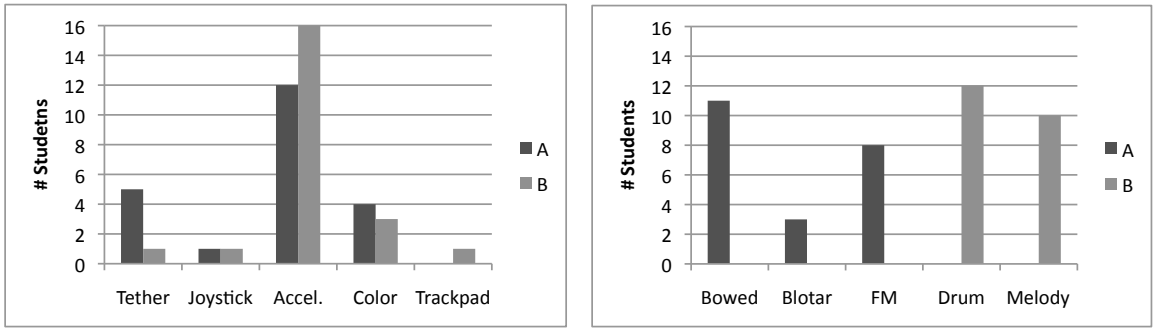
Students were provided with a special version of the Wekinator software that logged their actions as they completed Parts A and B. All actions such as training models, recording data, evaluating and running models, and changing algorithms and features were recorded to a file in real-time. The students emailed us the log files at the completion of the assignment. Students were aware that the logging was being performed in order to discover how they were using the Wekinator, and that the log contents would not be used in determining their grades.

All students completed the assignment, and all but one student provided the Wekinator log files. Eleven of those students completed Part A first and ten completed Part B first.

5.4 Observations

5.4.1 Interactive Systems Built

In the midterm assignment, all students successfully completed Part A and Part B, constructing an expressive continuous controller and a reliable gesture classifier. (Because one student was unsuccessful in retrieving the Wekinator log files, all logging analyses are performed over only 21 students.) Nearly all students used the synthesis



(a) Gestural Inputs

(b) Synthesis Algorithms

Figure 5.3: The number of students who chose each gestural input and synthesis algorithm, for Part A and Part B.

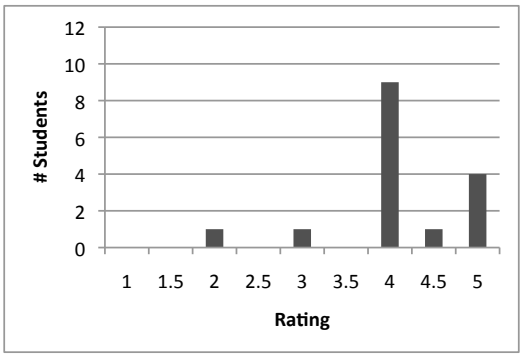
algorithms and gestural control inputs specified in the assignment instructions. Figure 5.3 shows the number of students who chose each input and synthesis algorithm for Part A and Part B.

Figure 5.4 shows students' levels of agreement with several statements regarding the success with which they were able to use the Wekinator in the assignment, rated on a 5-point Likert scale. In general, students were very successful in using the Wekinator to accomplish the goals of the assignment; the average agreement with all statements was above 4.0. Statement S1, "My model is musically expressive," received the lowest average score overall (4.1), and several students indicated in the written component of the assignment that they felt limited by the expressive potential of the synthesis algorithm that they chose, not by the Wekinator or their mappings.

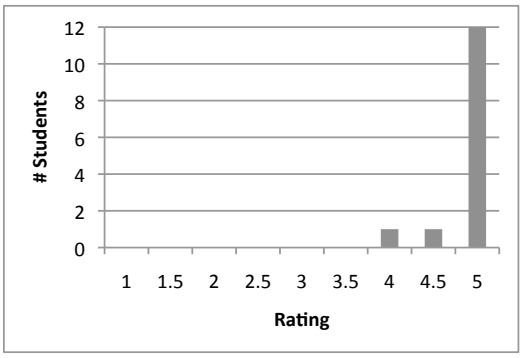
5.4.2 Actions Performed

In the midterm assignment, the logging data and students' written work clearly indicate that students employed an iterative approach to interactive model building, in which they retrained the model multiple times following changes to the training data or algorithm. Figure 5.5 shows the number of times a model was retrained over the course of each task, by each student (not counting the first time the model was trained). More than half of the 21 students retrained the model at least once, but there was great variation in how many times students retrained the models, up to a maximum of 17 times by one student in Part B. The average number of times a model was retrained in Part A was 3.3 (median = 1.0, $\sigma = 4.2$); the average number of times a model was retrained in Part B was 4.9 (median = 1.0, $\sigma = 5.9$).

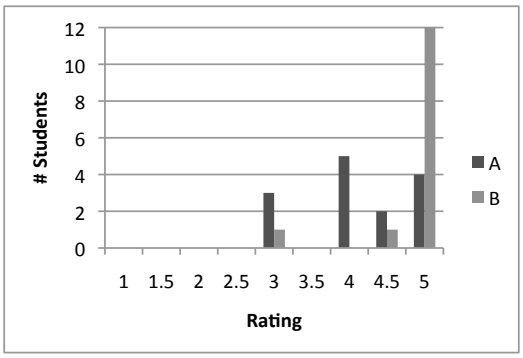
Between model trainings, students took one or more actions to modify the model, including modifying the training dataset and changing the learning algorithm or its parameters. In Part A, the model was modified by changing the dataset an average of 3.1 times (median = 1.0, $\sigma = 4.2$); the learning algorithm was never changed. In Part B, the model was modified by changing the dataset an average of 4.3 times (median = 1.0, $\sigma = 5.2$), and by changing the learning algorithm or its parameters an average



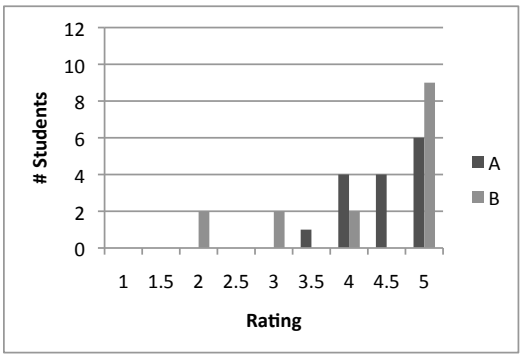
(a) Statement S1: "My model is musically expressive." (Part A only)



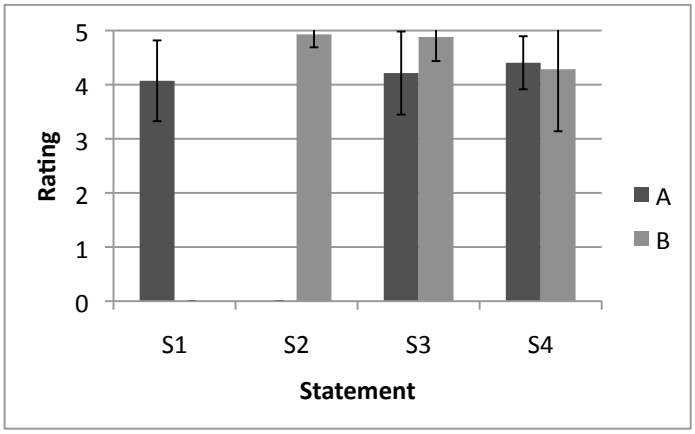
(b) S2: "My model provides reliable gesture classifications." (Part B only)



(c) S3: "I can reliably predict what sound my model will make for a given input gesture."



(d) S4: "Wekinator eventually learned what I wanted it to."



(e) Average and standard deviation of ratings for statements S1–S4 above.

Figure 5.4: Students' success against assignment criteria, self-rated as agreement with four statements on a 5-point Likert scale.

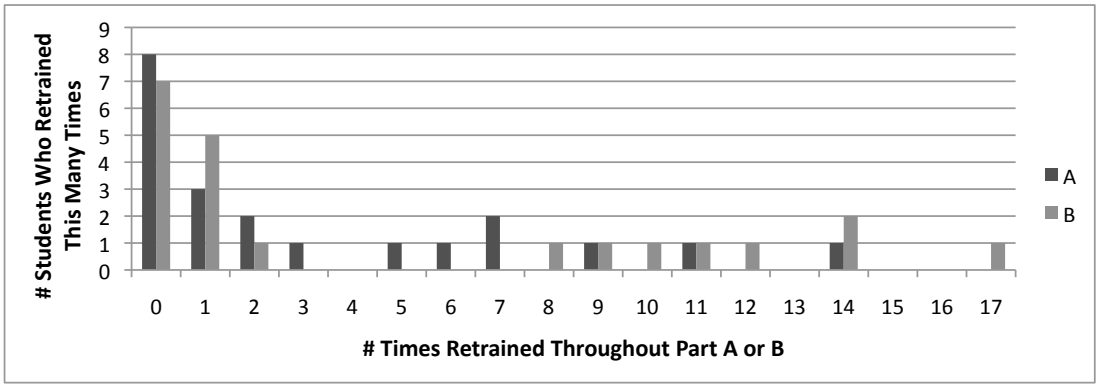


Figure 5.5: The number of students who retrained the model a given number of times, during Part A and Part B, not counting the first model training.

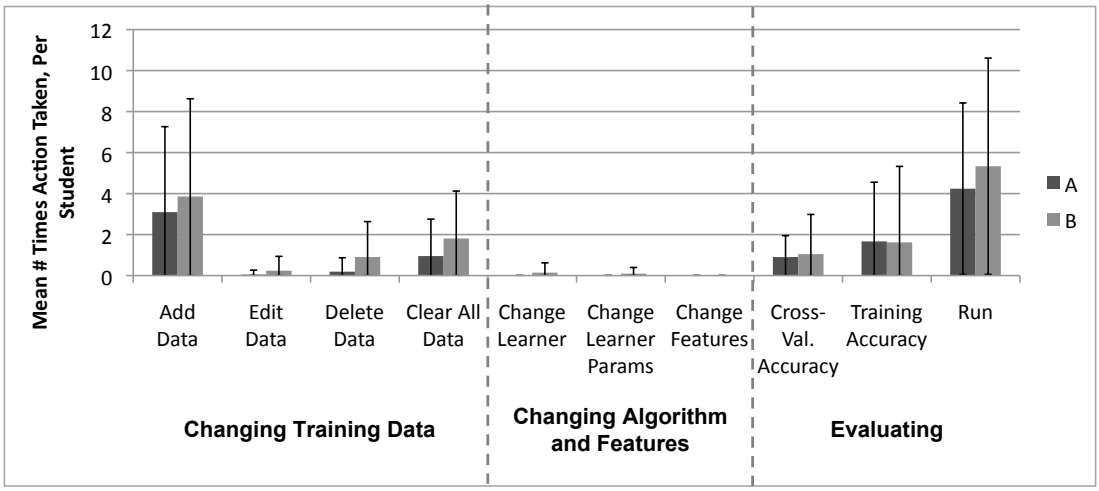


Figure 5.6: The average number of times a student took each action, in Parts A and B. Error bars show one standard deviation.

of 0.2 times (median = 0.0, $\sigma = 0.8$). The features used by the models were never changed in Part A or B, though one student changed the controller that she used in the middle of Part B.

Figure 5.6 shows how many times each particular action was taken in order to modify and evaluate a model, for Part A and Part B. Edits made to the training data and algorithms before the first model training are not included, and actions that were performed multiple times between consecutive model trainings (e.g., recording new training examples twice before retraining the model) are counted only once. Each student attempted to improve the model by modifying the training data significantly more frequently than by modifying the learning algorithm or its parameters ($p < .0001$ using a paired t-test), and each student evaluated the model by running it in real-time significantly more frequently than by computing cross-validation accuracy ($p < .0001$) or training accuracy ($p < .0001$).

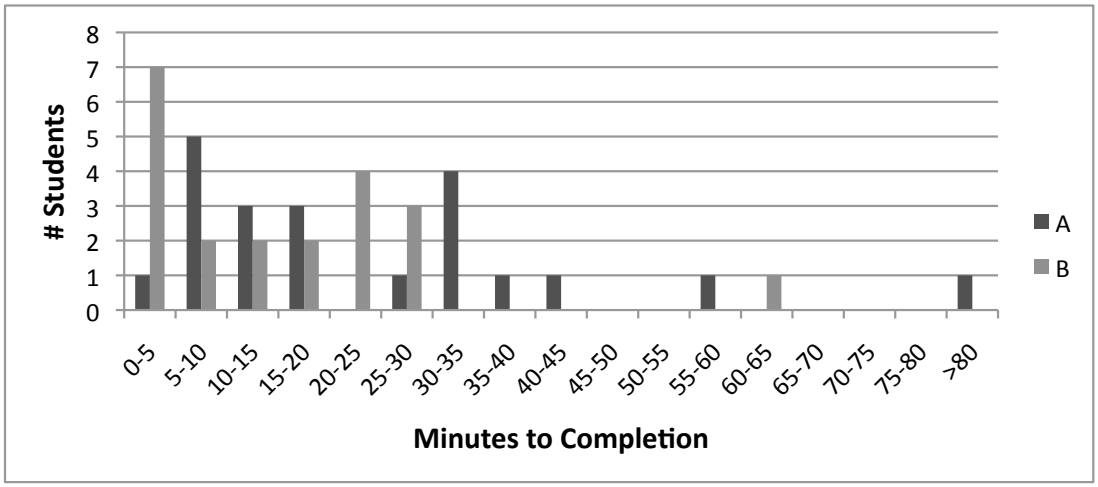


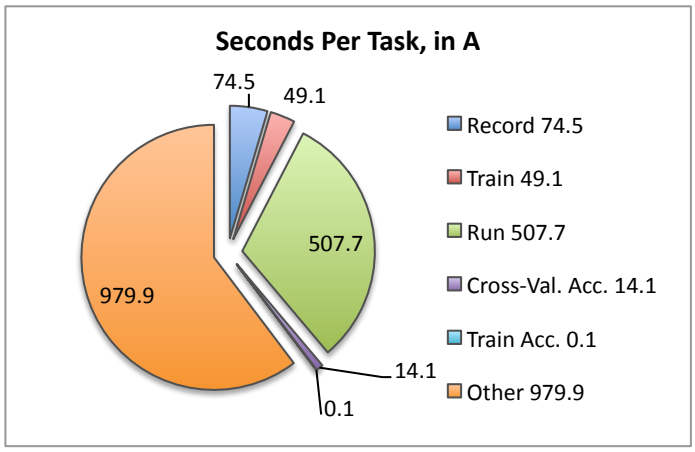
Figure 5.7: The total amount of time to complete Part A and Part B, by student.

Students created training data both using "playalong" recording (in which students set up a parameter "score" and recorded training data by gesturing along to the synthesizer playing this score; see Section 3.4.6) and by entering each set of parameter values in the GUI and demonstrating the gesture that corresponded to those values. While only 8 of the 21 students ever used playalong recording, playalong recording was used in 114 of the 188 total occurrences of students recording new training data during the assignment. Students never used the graphical data editor, perhaps because it was not demonstrated to them in class. Four students employed the spreadsheet editor to manually change feature or parameter values; three of these students made between one and three manual edits each, but one student made 95 manual edits as he attempted to fix his models' misclassifications in Part B.

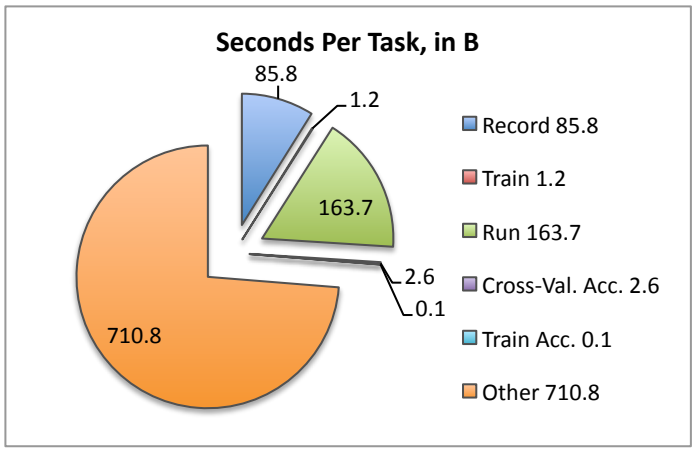
5.4.3 Interaction Over Time

Figure 5.7 shows the time taken by each student to complete each part of the assignment. All but two students took under 45 minutes to complete Part A, and all but one student took under 45 minutes to complete Part B. The average completion time was 27.1 minutes (median = 16.7, σ = 30.5) in Part A and 16.1 minutes (median = 14.6, σ = 14.3) in Part B. For most students, the Wekinator offered a reasonably fast way to build a reliable gesture classifier or an expressive continuous controller. Incidentally, the student who took the most time on a task, 145.1 minutes in Part A, did not appear to have had any particular trouble with the assignment; on the contrary, his written response (the longest of any student) indicates that he spent a lot of time experimenting with his blotar controller and exploring several different mapping strategies over the course of the task.

Figure 5.8 shows the average total time spent performing each action with the Wekinator, within Parts A and B. The action to which students devoted the most time was evaluating the model by running it, followed by recording new training data. The time spent to train the models and to compute cross-validation and training



(a) Part A



(b) Part B

Figure 5.8: The average number of seconds spent performing each action over the course of Part A or B, shown as a proportion of the average total task time.

accuracy was negligible. In the time spent on "Other" actions, the user was setting up the Wekinator for the assignment task, setting the synthesis parameters directly to test out new sounds, and likely reading the assignment instructions and performing other actions not loggable by the Wekinator.

The time needed to train the models was quite short: 11.5 seconds per training on average in A (median = 4.5, $\sigma = 19.4$) and 0.2 seconds on average in B (median = 0.1, $\sigma = 0.3$). A short training time—much shorter than the minutes or hours that are acceptable in many non-interactive supervised learning scenarios—is important in this context, so that training does not interrupt a user's interaction with the system and training time does not provide a disincentive to attempt to fix a model if the user is not satisfied with its behavior. The relatively fast training time can be attributed to the fact that training sets were generally quite small, compared to those used in other applied machine learning problems. The average training set size was 790.1 examples (median = 565.0, $\sigma = 604.7$) in Part A and 548.0 examples (median = 424.0,

$\sigma = 426.8$) in Part B. While the difference in each students' average training set size in Part A and Part B was not statistically significant ($p = .082$ using paired t-test), the difference in average training time was significant ($p < .05$). The difference in training time is due to both differences in learning algorithms and the number of synthesis parameters used in the two parts of the assignment: Part B required the training of only one classifier, while Part A required the training of between three and nine neural networks. We discuss the phenomenon of small training size further in Section 8.6.

5.4.4 Model-Building Strategy

Both the logging data discussed above and students' own written remarks on their strategies for model building emphasize an iterative process: students repeatedly built a model from the training data, evaluated it to assess whether they liked it, and attempted to improve it. In this section, we discuss in more detail the actions that students took and the criteria they used to evaluate models, and how they acted on the knowledge gained from model evaluation to attempt to improve the system.

Training Data Creation

A few students began Part A with quite clear ideas regarding the types of gestures and mappings that they wanted the model to learn. The students with a clear plan for how they wanted the instrument to work developed strategies for creating training examples that often focused on just one aspect of the mapping function at a time. For example, one student described his strategy: "I started by first determining which of the HID input parameters corresponded to which axis on the actual tether (i.e. HID_6 ... reflected the length of the right tether string). I then adjusted the neural networks accordingly so they would only be looking at what I wanted it to ... I then attempted to train each parameter independent of the others by unchecking their boxes and giving Wekinator some training samples and then training and running it. Based off how it did here, I went back and gave it more samples before moving onto the next parameter. After all three parameters were trained I went back and gave it more samples again for each (again, each parameter was trained independently of the others) before I was satisfied with how well it could predict what I wanted the parameters to be for a given motion."

Many students who did not have such clear a priori goals described a model-building strategy for Part A that was quite similar to the strategy employed by composers in Chapter 4: they created training examples matching sound parameters that they liked with a few different gestures, then let the model "fill in the blanks." One student wrote, "I found it the easiest and clearest to create a model from the outside in, sort of: specifically, I would create the extreme sounds and positions, train those extremes, and then fill in more information about the transition between the already learned extremes." Other students also indicated that they used iterative retraining to make a model gradually more complex, for example: "I started with

just experimenting with one of the parameters at a time, and slowly worked my way up so that eventually I was manipulating all three parameters."

In Part B, some students worked methodically to build the gesture classifier, for example carefully controlling the way in which parameters and gestures co-varied to clearly represent their ideas for how the classifier should work. As in part A, some students used iterative retraining to gradually increase the complexity of their model, adding new gestures and classes once a model was performing well. For example: "I didn't immediately do all of these gestures at first but started with only a few to get warmed up." Other students, though, provided training data for all gestures and classes all at once, or did not seem to have much of a conscious strategy.

Just as in Part A, there was a wide variety of among the extent to which students had clear plans about the sorts of models they wanted the Wekinator to build. In some cases, students' plans for which gestures they would use were informed by which gestures they thought the Wekinator would be able to easily classify. One student who was using color tracking to control the drum machine also formed her plan for the model based on how frequently she expected to use different classes in performance, writing: "I tried to assess which beats I would use more often and correlate them with [feature values] that were easier to obtain on the colour tracker."

In both Part A and Part B, students' strategies for creating and modifying the training data evolved as they worked with the Wekinator; we discuss this phenomenon further in Section 5.4.5.

Model Evaluation

The Wekinator allows users to evaluate trained models using two approaches: computing standard metrics (training and cross-validation accuracy) to assess the capability of the algorithm to accurately model the training set, and directly evaluating the trained model by running it on new gestural inputs in real-time and observing its behavior. In order to assess a newly trained model's performance and determine how to improve it, students employed cross-validation accuracy 13.8% of the time, training accuracy 28.9% of the time, and direct evaluation 93.7% of the time.

Over the course of the assignment, 13 students computed cross-validation accuracy at least once, and 12 students computed training accuracy at least once. Each student computed cross-validation accuracy an average of 0.9 times (median = 1.0, $\sigma = 1.0$) in A, and an average of 1.0 times (median = 0.0, $\sigma = 1.9$) in B. Each student computed training accuracy an average of 1.7 times (median = 0.0, $\sigma = 2.9$) in A and an average of 1.6 times (median = 0.0, $\sigma = 3.7$) in B. Students employed direct, hands-on evaluation significantly more often than they computed cross-validation or training accuracy, as discussed in Section 5.4.2; each student directly evaluated models an average of 4.2 times (median = 2.0, $\sigma = 4.2$) in A and an average of 5.3 times (median = 2.0, $\sigma = 5.3$) in B.

Students used accuracy metrics and direct evaluation to gain different types of knowledge about the models. In their written work, several students implied that they treated a high cross-validation or training accuracy as reliable evidence that a model was performing well, and students often reported cross-validation accuracy or

training accuracy scores of their final models as evidence to the grader that a model was in fact performing well or poorly. At least one student used cross-validation to validate his own model-building ability, writing "Following [dataset creation], I would usually quickly check the cross-validation and training accuracy and see if the Wekinator thought that my model was a good one. If it was, my next step was usually to run the model myself and observe how it reacted to different gestures." This echoes the findings of Amershi et al. (2010), who observed that users felt pressure to optimize cross-validation accuracy as an end in itself, rather than using it as an informative tool. Six students never computed cross-validation or training accuracy, and some students indicated that they did not understand the metrics or that they found them irrelevant or unhelpful: "...I preferred to just run the machine and see how it worked when I was actually trying to use it in practice."

Students reported using direct evaluation to assess models against a variety of criteria, encompassing correctness, musical expressiveness, complexity, and naturalness. In both Part A and Part B, students identified a model's behavior as incorrect when it produced an output contrary to what they believed was appropriate and expected. Students also used direct evaluation to assess the model against the assignment goals of musical expressiveness in Part A and reliably accurate classification in Part B. Like the composers in Chapter 4, students sometimes indicated that, in Part A, complexity and unexpected behavior were in fact desirable properties in a model. One student wrote: "I actually found myself happiest with my model when I had introduced less predictability into it," and indicated that his goal ended up being to build a model with "an optimal balance between controllability and versatility." In Part A and Part B, seven students mentioned that it was important to them that, when using the models, the gestures and gesture-sound relationships felt "intuitive" or "natural."

Model Modifications

Figure 5.6 on page 123 shows that the most frequent actions taken by students to improve or change models were the adding, deleting, and editing of training data. Students' written work reveals that they typically performed these actions for rational and predictable reasons. Students added new training data to correct errors or to make a model's behavior more complicated. They deleted subsets of examples to address particular problems in a model, for example deleting the last round of training examples recorded when a retrained model did not perform as anticipated, or deleting all training examples that affected a parameter whose model was not performing well. They cleared the training dataset when they gave up on fixing a model and wanted to start anew; one student wrote: "I found it often easier to just start over again when certain things were not working. When recording more examples, I often found that the training got even more muddled." Students who did change the learning algorithm changed it when they thought it should be possible to build a more accurate model from data.

In addition to modifying the training data to attempt to build models that better met their goals for the system, students also often modified the data to reflect changes to their goals, as we discuss next.

5.4.5 Human Learning and Adaptation

Changing Goals

Students often adapted their goals for what they wanted the Wekinator to learn based on what they discovered through direct evaluation. One reason for this adaptation was that, through experimentation with the Wekinator, they realized that it was too difficult or impossible to build a model that conformed to their initial ideas about how they wanted gestures to control sound. When this occurred, students sometimes simplified the learning problem, for example by reducing the number of classes in Part B or changing the gestures themselves to be more easily differentiable. One student actually changed from using the webcam color tracker to using the trackpad input, which she understood better, so that she could better control the learning problem she presented to the Wekinator.

Another reason that students changed their goals for the models was that, through hands-on experimentation with the system, they sometimes discovered models performing in unexpected ways that they actually liked better than how they had planned for the system to work. One student described this experience in Part B: "...[A]s I was trying to train [the model], I began to get a model that would play beats depending on the extremity of the tilt. Beat 0 would be stationary, beat 1 would then be a slight tilt in any direction, and beat 2 would be an extreme tilt in any direction. I hadn't really thought to do this initially, but once the training to some extent got there on its own I decided to go with it." Another student wrote in Part A: "First I wanted to make something that worked just how I wanted, but then I thought it more important to create something that was controllable but also generated awesome sounds I would never have thought of. In my estimation, this was a mission accomplished!"

Two students indicated that they chose control gestures to use solely through exploration, rather than starting from a set of gestures they were interested in the Wekinator recognizing. One student wrote, "I didnt have a very clear idea in my head of how I wanted the sound to change as the tether moved, so I used a trial and error method. Eventually, I realized that there were certain gestures that I wanted to correspond to certain sounds, so I recorded more examples to train the system accordingly. As I noticed patterns starting to emerge in my tests, I recorded more examples to make these patterns reliable while using the tether."

Through direct evaluation and retraining, students also learned about the tradeoffs associated with different variations of their models, and they used their judgement to pick a model with the set of tradeoffs that they liked the best. One student wrote, "I had originally tried to use all four beats. I first tried using a neutral position where the computer was not tilted at all, and later used a position where the computer was tilted to the left. I eventually decided to eliminate this fourth position, as every single time it did this, I would lose two or three of my other beats, and get a much less accurate model (usually a cross validation accuracy of 50%). I eventually forfeit the extra beat for a much more accurate model."

Learning Effective Strategies for Interactive Machine Learning

Students also adapted their strategies for effectively performing machine learning, based on their observations of how their actions affected the trained models. When asked about their strategies for model building, ten students indicated that they had learned during their interaction with the software to provide training data that more clearly expressed their intentions. One student wrote, "In collecting data, it is crucial, especially in Motion Sensor, that the positions recorded are exaggerated (i.e. tilt all the way, as opposed to only halfway)." Another wrote, "I tried to use very clear examples of contrast in colour for the tracker. If the examples I recorded had values that were not as satisfactory, I deleted them and rerecorded... until the model understood the difference." Many students reported that they looked for outliers in their training examples and deleted them. Some students even learned to balance class proportions in the training set ("Each extreme of a parameter should be trained with roughly the same number of examples"). This is remarkable, given that none of these machine learning concepts were introduced or discussed in class.

Learning Effective Gestures

Finally, students adapted the ways in which they interacted with the trained models, based on which gestures yielded the best results. Students' comments indicated that they did not necessarily view models' "musical expressivity" and "reliable classifications" as intrinsic properties of the models themselves; rather, these qualities had to do with the extent to which the *students* could learn to make musically expressive and reliably classifiable gestures *using* the models. Direct evaluation allowed students to find gestures that created the desired sonic outcomes, "practice" these gestures, and evaluate the extent to which a performer could learn to play the mappings well. Many of the students in the class were proficient musicians (outside of PLOrk), and their written comments revealed that they treated the process of learning to effectively play a model as similar to the process of practicing a traditional musical instrument, and that they viewed human effort and adaptation to be appropriate and necessary in both cases. For example, comments included: "The more time I spend with the instrument, the more I know how to control it" and "Practice is useful when learning an instrument."

5.4.6 Success in Using the Wekinator

As shown in Figure 5.4 on page 122, students agreed that they were able to successfully create musically expressive instruments and reliable gesture classifiers using the Wekinator. In general, the feedback from students in the written component of the assignment indicated that they enjoyed using the Wekinator and found it useful. Their comments included: "Learning by experimentation was a lot of fun!", "I very much enjoyed this project!", "I love Wekinator!", and "Its *so cool*, the Wekinator rocks." Furthermore, students were able to build models that they were happy with in just 27.1 minutes in A and 16.1 minutes in B, on average. This is an extremely

short amount of time compared to what would be required to build a reliable classifier or expressive continuous mapping by writing code in ChucK or Max/MSP, for example, even for an expert programmer.

A few aspects of the Wekinator were confusing or frustrating to some students. Some students did not realize that they could limit the features selected for each model or restrict the parameters affected by a training example, so they relied on encoding their goals for independencies among parameters and between features and parameters through carefully designing the training set (a difficult feat). Several students also expressed frustration that the learning algorithm they used did not create accurate models from the data. In particular, students using AdaBoost.M1 boosting on decision stumps encountered many difficulties creating accurate models for multi-class problems; these difficulties are predictable from the perspective of someone familiar with the algorithms, but students did not necessarily know that changing the algorithm would likely fix many of their problems. Apart from these issues, which can be addressed through interface improvements and better educating users about how to use the Wekinator, students did not present complaints or criticisms about the software.

5.4.7 Effects of Task and Order

To examine effects of task (A or B) and order (A first or B first) on the model building process, we conducted a two-way ANOVA with one within-subjects factor (task) and one between-subjects factor (order).[2] We examined the total number of model retrainings, the total time spent on the task, and students' agreement that they could reliably predict the sound the model would make for a given input gesture and that the Wekinator learned what they wanted it to (i.e., ratings for statements S3 and S4 in Figure 5.4). There was no significant effect of task on number of retrainings ($F_{1,19} = 1.08$, $p > .05$), total time ($F_{1,19} = 2.46$, $p > .05$), or S4 rating ($F_{1,19} = 0.21$, ns); there was a significant effect of task on S3 rating ($F_{1,19} = 12.04$, $p < .005$). There was no significant effect of order on number of retrainings ($F_{1,19} = 0.29$, ns), total time ($F_{1,19} = 1.46$, $p > .05$), S3 rating ($F_{1,19} = 1.19$, $p > .05$), or S4 rating ($F_{1,19} = 0.93$, ns). We discuss these findings in Section 5.5.2.

5.5 Discussion

5.5.1 The Wekinator as a Teaching Tool

As discussed above, students were generally very successful in achieving the assignment goals of building musical and reliable models, and they did so relatively quickly. In this regard, the Wekinator was a highly successful pedagogical tool: no other software system could have been used to allow students to build these types of musical

[2]Because the order was unbalanced with 21 students, we used SPSS to compute a Type III SS ANOVA.

Figure 5.9: Three PLOrk students performing their adaptation of George Gershwin's "Summertime" from *Porgy and Bess* during the midterm concert. The leftmost performer is playing the melody using laptop tilt, the middle player is using the Wekinator to control mandolin chords with the joystick, and the rightmost player is using the Gametrak tether controller with the Wekinator to play an FM-synthesis drone.

gestural classification and regression systems, without requiring them to have significantly more expertise in machine learning and requiring them to write a great deal of code. By enabling students to design their own classifier- and neural network-based systems very quickly, the Wekinator allowed students to experience new ways interacting with computer music systems, as well as to explore the musical consequences of their design decisions.

The Wekinator was a useful teaching tool in other assignments throughout the semester. All students used the knowledge gained in the assignment discussed in this chapter to create successful midterm performance pieces using the Wekinator. In the midterm performance, students worked in groups of two or three and built new interactive, gesturally-controlled instruments, then composed pieces for these instruments and performed them for their classmates and other Princeton students in a concert. Some of the more creative new instruments built using the Wekinator for the performance included a tether-controlled tubular bell synthesizer, a joystick-controlled mandolin chord generator in a laptop re-adaptation of Gershwin's "Summertime" (Figure 5.9), and an accelerometer-controlled algorithmic glockenspiel process, where the Wekinator affected the glockenspiel tempo and volume. Two groups of students also chose to use the Wekinator in the final course project, which was left open-ended with regard to the nature of the performance or installation that students created and the hardware and software tools they used. In one of these projects, pictured in Figure 5.10, the students created a "musical petting zoo" of instruments including a joystick harmony-generator and a "tether harp" that played notes when the strings were plucked. The students invited bystanders to play with these instruments, then taught groups of people to collaboratively perform a simple piece.

Figure 5.10: The "Musical Petting Zoo" final course project in action. These visitors to the project exhibition are learning to play a Wekinator-based joystick harmony generator and a Gametrak tether harp.

In addition to providing a tool for students to effectively build new instruments and installations, several students commented in their written work later in the semester that they found the Wekinator useful as a creative tool for discovering and realizing new uses of the computer in their projects and compositions. For example, excerpts from the midterm performance reports include: "Generated mappings can produce interesting and unique sounds that you never would have thought of on your own, which can later be used to great effect," and "Sometimes, too, interesting things that you didn't intend happen 'between the cracks' of the examples you trained the model with. This can lead to some neat musical effects."

The Wekinator was used throughout the semester as a tool for teaching modular design, object-oriented programming, and signal processing. In the midterm assignment, several groups of students chose to implement their own synthesis classes. By implementing the Wekinator ChucK synthesis class interface (see Section 3.3.4), they were able to use the Wekinator to control new synthesis algorithms and compositional processes that they designed themselves. Also, in an assignment later in the semester, all students used their signal processing knowledge gained in the course to build a ChucK audio feature extractor that implemented the custom ChucK feature extractor interface (see Section 3.3.2).

Involving the students in designing their own code components to plug into the Wekinator not only allowed them to discover through practice why modular design and APIs are useful; it also gave them a quick way to experiment with their own feature extractors and synthesis algorithms, discover whether or not they worked correctly, and create something musically interesting with them without writing any additional code. Because the Wekinator was used so extensively in the course, some students expressed that they felt limited by the small number of available, pre-written synthesis algorithms available for them to use. Therefore, in future classroom use of

the Wekinator, we plan to engage students themselves in building more synthesis classes and making them available to other students and all Wekinator users in an online repository.

As machine learning was not itself a primary focus of the course, we did not directly evaluate how much students learned about machine learning during the assignment or during their use of the Wekinator. However, as we discussed above, some students did learn a surprising number of fundamental and subtle machine learning concepts, including the circumstances under which some algorithms worked better than others, and how properties of the training data such as noise, inter-class and intra-class variability, and class imbalances affected the trained models. They learned this information not from discussion in class, but simply by interacting with the system and observing how different interactions and datasets produced different models. Therefore, using interactive machine learning seems like a potentially useful and fun way to explicitly teach students about machine learning principles and algorithms, and this is another potential future application of the Wekinator.

5.5.2 Interactive Supervised Learning and Task Type

As discussed in Section 5.4.7 on page 131, students' level of agreement with Statement S3 was the only measured quantity where a significant difference was found between Part A and Part B: students more strongly agreed in Part B than in Part A that they could reliably predict the sound the model would make for a given gesture. According to the students' written responses, some of them intentionally created models with some built-in unpredictability in Part A, because they felt such models were more interesting or musical. In Part B, on the other hand, unpredictability seemed to be in competition with the primary goal of the task, creating a reliable gesture classifier. Future work might more deeply investigate the differences in interactive and algorithmic needs of users with different priorities regarding predictability versus reliability.

Based on the students' written discussion of their strategy and evaluation methods, we believe there were other important differences between the way students built and evaluated the systems, and perhaps differences in how to improve the Wekinator to better support different types of tasks. The extent to which students had clear and fixed ideas about the gestures and gesture-sound relationships they wanted to learn did seem to be important in determining the strategy that they used to build the models. Several students with clear goals for the models went to great lengths to carefully design the training dataset to represent the precise gesture-sound relationship that they wanted, and it was students in Part B with clear ideas about the mapping they wanted who changed the learning algorithms or expressed frustration that the classifier they were using did not work. While more students in Part B than Part A expressed having clear ideas about the mapping they wanted before starting to use the Wekinator (19 in B, 8 in A), many students in B ended up changing those ideas once they started working with the system; so in this study, the extent to which students had fixed ideas they wanted the Wekinator to learn did not break down cleanly by task. Students' written responses regarding the extent to which

their strategy remained fixed were not clear enough to allow testing for significant differences in their behaviors. Additionally, it is unclear why different students had such different goals for the system: this could be a matter of demographics (e.g., their academic major), aesthetic priorities, or other factors.

We believe that future work might explicitly compare differences in how users execute the interactive building of models when they have fixed versus more flexible goals, and future work on improving interfaces for interactive machine learning might consider the needs of these types of users separately. For example, an interface that can provide feedback to a user indicating that a different learning algorithm might be more appropriate for the current training dataset might be more valuable to a user with fixed goals who has more limited ability to change the training set. We discuss these ideas further in Chapter 6, where we present work with a user whose goals for the trained models are relatively fixed by musical conventions, and in Chapter 8.

5.5.3 Improving the Wekinator

Students' written comments highlighted several aspects of the Wekinator that might be improved to make it more usable by people who are machine learning novices. First, as mentioned above, several students indicated frustration that their models were incapable of learning more than two or three classes. Examining the logs of their work, it became clear that these students were using AdaBoost.M1 boosting on decision stumps. This algorithm will perform poorly in certain simple classification problems, such as recognizing a laptop's tilt forward, backward, left, and right. It seems that the students used AdaBoost.M1 because it was the first learning algorithm in the alphabetically-organized list in the Wekinator GUI, and they used decision stump base learners because that was the default setting for the AdaBoost.M1 algorithm. In response, we modified the Wekinator so that the default setting for AdaBoost.M1 is now to boost on decision trees, which do not suffer from the limitations of decision stumps on multi-class problems. The question still remains, though, of how to most effectively educate novice users about which algorithms might perform best for their data, or to inform them about how they might change the parameters of an algorithm to achieve better results. At the very least, offering the user the suggestion that he might experiment with changing the learning algorithm if he is having problems might circumvent some of the frustration experienced by the students.

The question of how to educate novice users about certain machine learning concepts that might be helpful to them is also relevant to other aspects of the system, including interfaces for feature selection, adding meta-features, and computing training and cross-validation accuracy. It was clear from students' comments that some of them did not understand how to use or interpret one or more of these components of the Wekinator. However, we believe that machine learning novice users could be better educated about these concepts through careful explanations within the user interface and/or a "help" system or tutorial, and that these users should be able to use the Wekinator effectively for many problems without having to acquire too much theoretical knowledge about machine learning. In the near future, we hope to collab-

orate further with target novice users, such as students and professional composers, to create more helpful documentation.

5.5.4 Further Discussion

We further discuss the implications of this work regarding algorithm and interface design, the use of interactive machine learning by novices and in creative work, and the larger role of interaction in supervised learning in Chapters 8 and 9.

5.6 Conclusions and Future Work

In this work, we have found that the Wekinator enables students to successfully and quickly build musically expressive and accurate models, and observed that students relied greatly on being able to evaluate models in a hands-on way and modify models' training data in order to assess models' subjective quality and improve them over time. We have also gained a valuable perspective on how the Wekinator can teach users to develop effective strategies for interactive machine learning, and how interaction can over time help users develop and change their goals for the interactive systems they are creating, as well as allow them to practice interacting effectively with the models they have built.

This work suggests several veins of further research, including studying how the flexibility of users' goals for supervised learning might impact their interface and algorithm requirements, how to better support interactive machine learning by users who are machine learning novices, and how to integrate hands-on interaction into the teaching of machine learning concepts. This work also suggested several improvements to the Wekinator, some of which have been implemented and some of which are planned for the near future.

Although the Wekinator was not developed as a teaching tool, our experiences using it in the classroom have been remarkably positive: students were able to use the Wekinator to create projects that would not have been possible otherwise, and they were able to learn about a variety of topics, including machine learning, object-oriented programming, sound synthesis, and gestural analysis in a hands-on manner that engaged their creativity. At the same time, observing students using the Wekinator and hearing their feedback about the software was highly informative to us, not only suggesting how the software could be improved, but also suggesting new research questions. In our future work with the Wekinator, we hope to continue to create and take advantage of these synergies between teaching and research.

Chapter 6

Bow Gesture Recognition

6.1 Introduction

In this chapter, we discuss work undertaken with a professional cellist/composer to build a gesture recognition system for a commercially-produced, sensor-equipped cello bow. This bow, called the "K-Bow," contains embedded sensors for measuring the position and motion of the bow in real-time (McMillen 2008). One goal of this work was to build a gesture classification system for standard cello bowing gestures, such as bow direction ("up-bow" or "down-bow") and articulation (e.g., "legato," "marcato," and "spiccato"), for use in composition and live performance. A second goal of this work was to investigate how interactive machine learning can be useful for building gesture classifiers for discriminating among a pre-defined set of musical gestures. We sought to discover the subjective criteria employed by the cellist to evaluate trained models, how interactive model evaluation and training data editing enabled her to improve the models, and the ways that she was educated and influenced by her interactions with the software.

In this work, we have employed a user-centered design approach to the collaborative creation of the classifier suite and supplemental software infrastructure, in conjunction with an observation-analysis approach to discovering how the cellist employed the Wekinator to evaluate models, create training data, and refine her strategies for effective model-building. We begin this chapter with an overview of related work on bowing gesture analysis, a description of the K-Bow, and a more thorough discussion of our research motivation and goals. We describe our research method in detail and present the findings of our collaboration with and observation of the cellist/composer. Specifically, we present our observations regarding how interactions with the Wekinator were used in model-building, the quality of the models produced, the cellist's techniques and criteria for model evaluation, the ways in which the Wekinator drove the cellist's own learning and adaptation, and the cellist's overall evaluation of the Wekinator and models. We then briefly discuss the findings of this work with regard to the subjective and objective evaluation of supervised learning systems, the efficacy of applying interactive supervised learning with the Wekinator to bow gesture classification, and proposed future improvements to the Wekinator software. We

further discuss the implications of this work regarding interactive machine learning applications, interfaces, algorithms, and evaluation methods in Chapter 8.

In the work presented in this chapter, we found that the cellist was able to successfully employ the Wekinator to create bow gesture models that were of a sufficiently high quality to use in performance. She employed an interactive, iterative approach to building most of these models, in which she alternated between evaluating models and taking action to improve them by changing the training set, feature selection, or learning algorithm. We found that the cellist employed a variety of criteria for assessing the quality of a trained model, beyond just an assessment of its correctness; furthermore, the cellist's subjective rating of a model's quality did not always positively correlate with its cross-validation accuracy. These findings raise questions regarding the extent to which cross-validation and other accuracy measures are useful evaluation metrics in interactive contexts. We also found that interacting with the Wekinator led the cellist to become a more effective user of interactive machine learning, enabled her to redefine her goals for the models over time to account for what the models were able to learn well, and led her to gain a new perspective on her own bowing technique.

6.2 Background and Motivation

6.2.1 Bowing Gesture Classification

The problem of identifying and analyzing standard violin bow strokes by applying supervised learning to analyze bow sensor data has been studied previously (Peiper et al. 2003; Rasamimanana et al. 2005; Young 2008). This prior work focuses specifically on classifying different bowing articulation techniques, where articulation is defined as the "... manner in which notes are joined one to another by the performer; specifically, the art of clear enunciation in singing and precise rhythmic accentuation in instrumental playing..." (Slonimsky 1998, 16). There are several standard string instrument articulation techniques, each of which prescribes a particular use of the bow and bow arm before, during, and after contact with the strings; definitions and instructions for producing each articulation can be found in Flesch (2000).

Peiper et al. (2003) built a bow position sensor system using a pair of electromagnetic field sensors, then studied the application of decision tree classifiers to discriminate among subsets of standard bow strokes (martelé, détaché, spiccato, legato, and staccato). They designed the feature extraction system so that each feature vector characterized the bow position, speed, and acceleration for a complete bowing action in a single direction (i.e., an up-bow or a down-bow). The dataset for this work was collected by recording sensor outputs during a human demonstration of different bow strokes, then manually annotating the dataset with the proper bow labels; the design of the dataset (e.g., the proportions of each class, and the string or dynamic level used to demonstrate each stroke) is not otherwise specified in the published work. The decision trees obtained accuracy in the range of 71% (discriminating among all

five bowing classes) to 100% (discriminating between only détaché and martelé) on the data.

Rasamimanana et al. (2005) more thoroughly investigated the effects of features, performer, dynamic level, and tempo on the classification of martelé, détaché, and spiccato strokes. For the measurements, they attached position and acceleration sensors to a standard violin bow. The data used for experimentation was collected from two violinists who were directed to play each articulation while varying the violin string, dynamic level, and tempo in a prescribed manner. It was observed that the maximum and minimum acceleration and velocity in the direction of the bow movement were the features most predictive of the articulation style. In applying a k-nearest neighbor algorithm to the three-stroke classification problem on the data collected from the two performers, Rasamimanana et al. achieved per-class classification accuracy in the range of 85.8% to 96.7% on a held-out test set.

Most recently, Young (2008) applied a k-nearest neighbor algorithm to discriminating among six articulations: accented détaché, détaché lancé, louré, martelé, staccato, and spiccato. The bow used in this work was a standard violin bow specially outfitted with sensors for measuring downward and lateral force, three axes of acceleration, and angular velocity around these three axes. Data was collected from eight violinists playing an musical excerpt demonstrating each articulation on every string, and playing at a single dynamic level and a fixed tempo set by a metronome. In this work, each data instance used for training and classification was comprised of features extracted over a set of sequential, tempo-controlled bow strokes of the same type; that is, the goal of the classification was not to provide a note-level or instantaneous classification of articulation. Applying a standard dimensionality reduction technique to the dataset followed by a k-nearest neighbor classifier resulted in per-class classification accuracies of 91.7% to 97.9%, computed by three-fold cross-validation.

The dataset creation and evaluation methodologies employed in this prior work, especially in the work by Rasamimanana et al. and Young, have treated the problem of bow gesture classification as a conventional machine learning problem. The datasets were carefully designed, with certain bowing characteristics (such as tempo) intentionally held fixed in order to create a simpler problem than that of classifying realistic performance-time gestures, while other bowing characteristics (such as which of the four instrument strings were played) were intentionally varied in order to create classifiers that were robust to these variations. Having created a dataset that was thus representative of the researchers' chosen scope of the problem, the goal of algorithm and feature selection was to model the dataset as faithfully as possible, and to measure success using test set or cross-validation accuracy. Therefore, the evaluations performed in this prior work are not directly informative of how accurately the classifiers would classify bow gestures used in an actual performance, nor how useful they would be for a composer desiring to incorporate them into a composition. However, these results do indicate that supervised learning is a promising technique for classifying articulatory bow gestures from these types of sensors and features.

6.2.2 The K-Bow

The K-Bow is the first commercially-developed, mass-produced sensor bow for string players (McMillen 2008). It contains several sensors, shown in Figure 6.1, for measuring the position and motion of the bow in real-time. A three-axis accelerometer located inside the frog (i.e., the large, rectangular assembly at the bottom left of Figure 6.1, located at the end of the bow held by the player) senses tilt and acceleration of the bow in space. A grip sensor senses changes in the grip pressure and surface area of the cellist's bow hand. An angle-sensitive pressure sensor located at the junction between the bow hair and the frog measures changes in the tension of the bow hair as the cellist plays the strings of the instrument. The player also affixes a small circuit board, shown in in Figure 6.2, beneath the fingerboard of the instrument. This board creates an RF field and an infrared modulated wide field light cone, whose interactions with the loop antennas inside the bow stick and with the infrared detector inside the frog allow the measurement of the bow position and angle relative to the instrument. These sensors are summarized in Table 6.1.

The K-Bow is manufactured in versions for violin, viola, cello, and bass. Each version of the bow is designed to allow the string player to play using standard technique without encumbrance by the sensors, and this is accomplished through a wireless setup and through the bow's physical construction, which is designed so that the size and distribution of weight throughout the bow match a standard instrument bow. The power source and circuitry for the on-bow sensors are located inside the frog, and the sensor values are wirelessly communicated to a computer up to 10m away via Bluetooth. The published data rate for the Bluetooth transmission or sensor values is up to 625Hz, and we observed similar data rates in our work.

The K-Bow is shipped with a software suite, K-Apps, which receives sensor values from the bow. This software provides a GUI interface for sensor calibration and debugging, for example to allow the musician to check that the Bluetooth connection is still alive and all bow sensors are working properly. K-Apps performs real-time scaling of sensor values (e.g., into the range 0–4095), and it provides infrastructure for sending sensor values to other software programs via OSC or MIDI, as well as mapping sensor values directly to controlling K-Apps' built-in modules for audio spatialization, sample looping, and other musical processes. In our work, we used K-Apps only for sensor calibration and scaling, debugging, and sending sensor values to our own software via OSC.

6.2.3 Motivation and Research Goals

One goal of this work was to use the Wekinator to construct a set of robust classifiers for standard cello bowing gestures that the cellist/composer could use in composition and performance. As a professional computer music composer, she often creates interactive computer music compositions in which human actions trigger or dynamically influence sound and visuals produced by the computer. Additionally, she often participates in the performance of her own compositions, playing the cello with the K-Bow as well as as manipulating the computer directly through a GUI. Having access to a

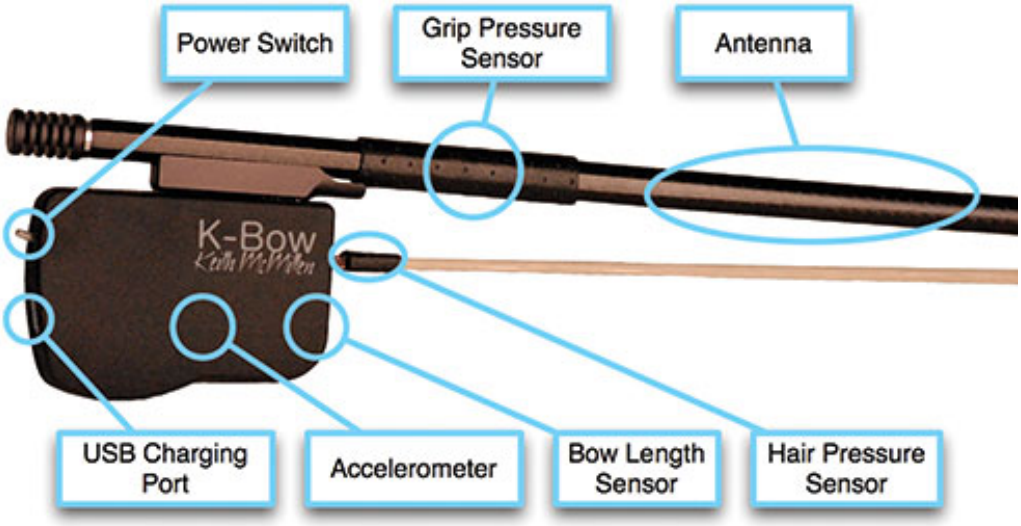


Figure 6.1: The K-Bow sensors. From Keith McMillen Instruments (2009b).

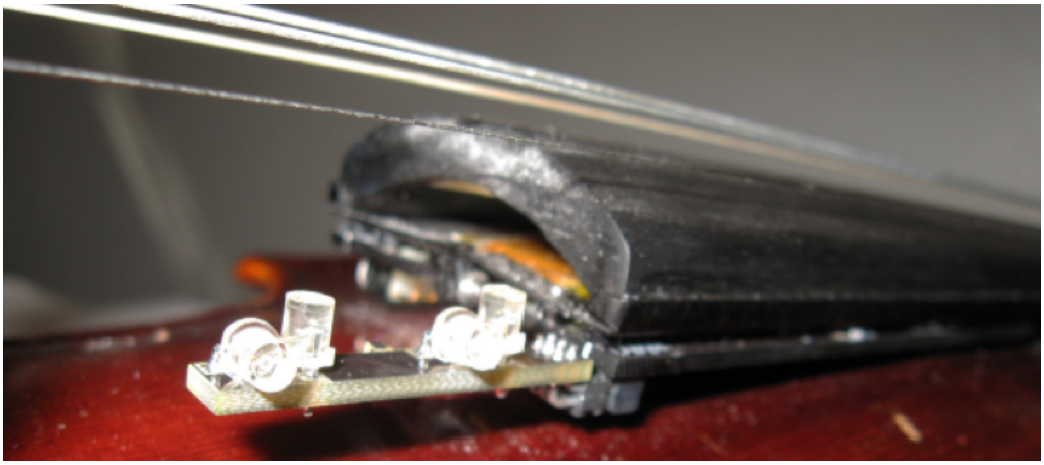


Figure 6.2: The K-Bow emitter, attached beneath the instrument fingerboard. From Keith McMillen Instruments (2009a).

set of real-time bow gesture classifiers would enable more natural performance-time interactions with the computer; in the words of the cellist, "It allows me to augment the bowing skills I spent years working on." These classifiers could enable her to trigger and control computer processes using her existing repertoire of bowing techniques, and potentially enable the computer to track the position of the cellist as she plays through a notated score. Additionally, bow gesture classifiers can reshape the process of composition, allowing her to focus on how the musical meaning of bowing gestures—rather than the sensor values themselves—might affect the evolution of a piece. This presents clear practical benefits: "The Wekinator makes the composi-

Table 6.1: A summary of K-Bow sensors.

Name	Description
x y z	Bow acceleration and tilt, measured by internal 3-axis accelerometer
hair (h)	Bow hair tension
grip (g)	Grip pressure and surface area
length (l)	Horizontal distance between frog and fingerboard
bridge (b)	Vertical distance between bow and bridge
tilt (t)	Tilt of bow relative to instrument

tional/programming process faster—its something I've wanted to do for a long time, but the amount of data was daunting." Finally, incorporating computer understanding of bowing gestures also has aesthetic appeal to her as a composer, as "the bow is really where the expression in a string instrument lies."

A second goal of this work was to investigate human-computer interaction with supervised learning algorithms in a gesture classification task that was more constrained than those explored in the previous two chapters. While the composers in Chapter 4 and the students in Chapter 5 had some leeway in choosing and changing the gestural vocabulary and the gesture-sound relationships they wanted the Wekinator models to learn, the learning problems addressed in our work with the K-Bow were constrained to modeling a gestural vocabulary that was fixed according to musical conventions. Furthermore, while the users studied in the previous two chapters were able to adapt their own performance-time gestures in order to elicit certain model behaviors, the goal in this work was to create classifiers that worked with the cellist's existing bowing technique, without any adaptation on her part. The aspects of human interaction that we were most interested in were the evaluation criteria and techniques used by the cellist to evaluate bowing gesture models, the ways that interaction with the Wekinator enabled her to improve the models, and the ways that the cellist was educated and influenced by the interactive machine learning process.

This work complements previous research on bowing gesture classification in that it is focused on the process and techniques through which people—perhaps the performers themselves—can build gesture classifiers that are most accurate and useful. Prior work has focused on designing and evaluating supervised learning systems that model a carefully pre-defined dataset as accurately as possible, without attention to how the models will be used in practice, and without engaging musicians' interaction or expertise beyond the initial creation of the fixed training set. In contrast, the goal here is to enable the musician who will be performing with the models to apply her musical expertise to making them as useful as possible. As we will demonstrate, this entails both enabling her to evaluate the models based on her own criteria for usefulness (which, as it turns out, includes more than just classification accuracy), and allowing her to take action to improve the models against these criteria.

6.3 Preliminary Project

We conducted a preliminary project with the cellist/composer and with the software engineer of K-Apps to develop and refine infrastructure for computing features from the raw bow sensor outputs and communicating these features to the Wekinator. In parallel, we collaborated with the cellist to teach her how to use the Wekinator, solicit her feedback on its user interface and design, and experimentally apply it to classifying several standard bowing gestures. These classification problems, which are further explained in Table 6.2, included bow direction, position on or off the strings, speed, vertical position, horizontal position, roll, and articulation. We aimed to demonstrate that the standard classification algorithms and interactive supervised learning process supported by the Wekinator—in particular, the use of training sets created through a few minutes of unstructured, interactive demonstration—were sufficient to allow a K-Bow user to construct her own working classifiers. We also wanted to discover which methods for segmenting and extracting features from the sensor outputs were viable, given that previous work on bow gesture classification had employed different sensors and a variety of segmentation methods. Additionally, we sought to discover which improvements to the Wekinator software were necessary to better support this type of classification task, and to implement them. This work was presented at the Third International Conference on Music and Gesture (Fiebrink, Schedel, and Threw 2010).

Our working process in this preliminary project was unstructured and exploratory. We met with the cellist in person on four occasions, for several hours each time, over the course of six months. During each meeting, we showed the composer how to use the most current version of the Wekinator, tested our current feature extraction and communication software, experimented with building gesture classifiers using different algorithms and features, and discussed how the Wekinator might be improved to make the classifier-building process easier. In between meetings, we further developed the Wekinator and feature extraction software based on our experiences and discussion.

We were successful in building proof-of-concept classifiers for each of the seven bow gesture classes listed above. The cellist did not rigorously evaluate the classifiers, but in each of the seven problems, she assessed that the classifier's accuracy was adequate and that she was confident she could address remaining significant errors in its performance through further refinement of the training data or algorithms within the Wekinator. Computing cross-validation accuracy of these classifiers yielded scores in the range of 80% to 100%.

This project led to several features being added to the Wekinator software. Most notably, the graphical interface discussed in Section 3.4.5 was created for visualizing and editing the training data. By using this interface to manually add class labels, the cellist could use more natural performative gestures to create the training data, switching fluidly between gesture classes (e.g., up-bows and down-bows) without pausing to interact with the GUI. This interface was also useful in cleaning up the training data, for example removing ambiguous instances recorded while the cellist was switching from an up-bow to a down-bow.

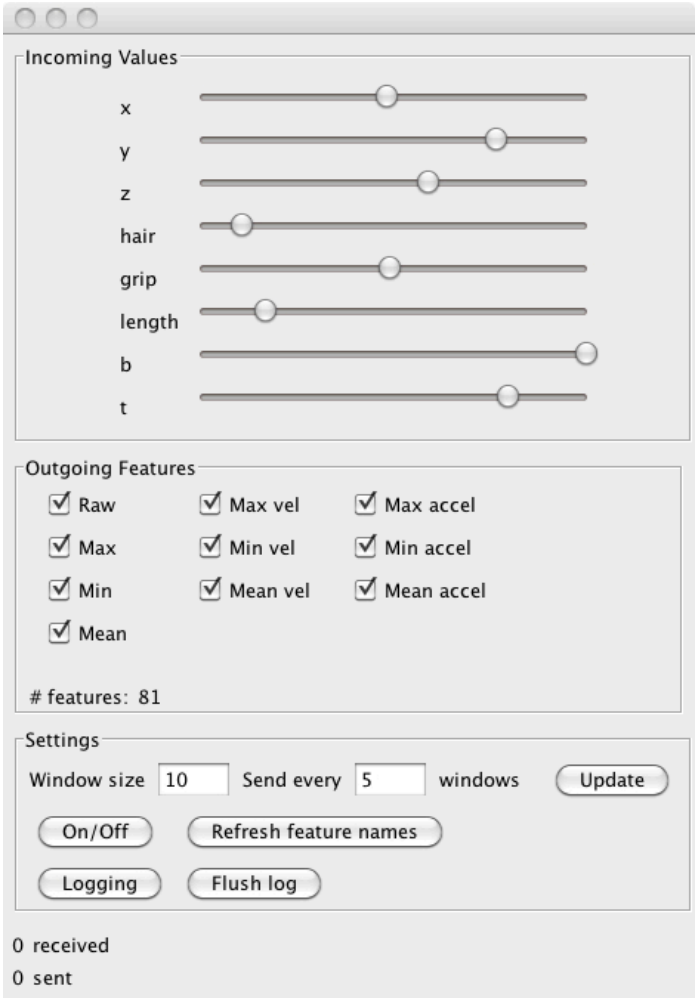


Figure 6.3: The K-Bow feature extraction GUI.

This project also led to the development of a K-Bow feature extraction application, external to the Wekinator, which computed features found to be useful for bow gesture classification. This application's GUI is shown in Figure 6.3. The values of the eight bow sensors are sent to this application from K-Apps, and it computes the feature vector and sends it to the Wekinator using OSC. For each of the bow sensors, this application computes the average, minimum, and maximum of the sensor value, and of the first- and second-order differences of the sensor value, calculated over a sliding analysis window whose size is specified by the user. The software also samples the raw sensor values once per window. The GUI allows a user to select which of these available features are sent to the Wekinator, adjust the window size, and adjust the rate at which features are computed and sent (i.e., the "hop size"). Furthermore, to aid in debugging, the GUI displays the current value of each K-Bow sensor output.

Table 6.2: Bow gesture classification tasks.

Name	Description
Direction	The direction the player is bowing, e.g., up-bow, down-bow, neither
On/Off String	Whether or not the bow is in contact with one or more strings of the instrument
Grip	Whether or not the cellist was squeezing the grip sensor
Roll	Whether or not the bow was rolled with the edge of the hair against the strings (to play more quietly) or positioned normally, with the hair more flat against the strings
Horizontal Position	The horizontal position of the bow relative to the instrument, i.e., whether the frog, middle, or tip of the bow is in contact with the strings
Vertical Position	The vertical position of the bow relative to the instrument, i.e., sul tasto (bow over the fingerboard), sul ponticello (bow near the bridge), or neither (bow in the middle)
Speed	The speed of the bow against the strings, e.g., "Very slow" to "Very fast," according to the cellist's own definitions of these terms
Articulation	The bowing technique employed to affect notes' onsets, releases, and transitions, including: legato (smooth and connected), marcato (onsets emphasized and slightly detached), spiccato ("enunciated" and percussive), riccocet (a "bouncing series of rapid notes"), battuto (struck with the wood of the bow), hooked (re-articulation of notes without a change in bow direction), and tremolo (rapid alternation of up-bows and down-bows) (see Flesch (2000) for further discussion)

6.4 Method

Having established the Wekinator as a suitable tool for building bow gesture classifiers, we conducted a more formal study of the application of interactive supervised learning to the construction of performance-ready classifiers. In this study, conducted five months after the preliminary project, we sought to address our research questions through observing and analyzing the cellist's actions and comments as she worked to construct eight usable classifiers for the gestures in Table 6.2.

The process of applying the Wekinator to creating these classifiers was collaborative. We assisted the cellist with the machine learning component of the work, discussing the algorithms, parameters, and features with her. She was responsible for assessing the quality of the trained classifiers and creating training data, and we did not assist in these tasks. We recorded the cellist's actions and comments throughout this process using a combination of written notes, video, and automated logging. The Wekinator logged all interactions with the software, and it saved all models, all train-

ing data, and all data generated as the cellist ran trained models to classify new bow gesture inputs. Additionally, we asked the cellist to evaluate and verbally rate the quality of each trained model on a 10-point scale. We recorded these ratings along with her commentary explaining the scores, if any.

For each of the eight gesture classification tasks, the cellist started from an empty training set and built a classifier from scratch (i.e., the training data and models from the preliminary study were not used). She worked on improving the classifier for each task until she was satisfied with its performance or felt she could not improve it further. The cellist freely chose the order in which the classification tasks were addressed and the amount of time to allocate to each one. A secondary set of classifiers were built in a separate, later session of model-building for five of the more difficult tasks, in order to allow the cellist to use her knowledge gained from building the first set of models. For clarity in the following discussion, we will refer to the first session of model-building, in which models were built for all eight classification problems, as "A," and to the second session, in which models were re-built from scratch for five classification problems, as "B."

6.5 Observations

6.5.1 Interactions with the Wekinator

In building the Vertical Position, Grip, and Roll classifiers in Session A, the cellist did not employ an iterative approach to model building. After creating each classifier and directly evaluating it, she rated their subjective performance as "10" and judged that they did not require further improvement. These three classification problems were among the most straightforward of the tasks: each had only two or three classes, which were easy for the cellist to demonstrate unambiguously and which depended on quite simple properties of the K-Bow sensor features. Iteration was not needed because the initial set of training examples clearly represented the problem, and the initially-chosen classifier algorithm was able to create an accurate model from the data.

In building the other five classifiers in Session A, the cellist employed an iterative approach to model building. She identified problems with a model through evaluating it (typically by running it on new gestures in real-time, as discussed in Section 6.5.3), then addressed these problems through modifications of the training data, algorithm, and/or features. Figure 6.4 illustrates the actions taken to build the models in the five classification tasks in A for which the model was trained more than once. These actions are summarized in Figure 6.6a, which displays the total and per-task average of the number of times each system interaction occurred.

In building each of the eight classifiers in Session A, the learning algorithm was retrained an average of 3.7 times (σ = 6.8, median = 1.0) (not counting the first training). Each retraining followed a modification of the training set, learning algorithm, and/or selected features, and each of these actions was important in at least one task. Among the five tasks for which the classifier was retrained at least once,

the training data was modified somehow after its initial creation an average of 2.5 times (median= 1.5, σ = 3.1), the learning algorithm and/or its parameters were changed an average of 2.8 times (median = 1.5, σ = 4.1), and the selected features were changed an average of 2.3 times (median= 1.0, σ = 3.8).

In the second session of model building, B, the learning algorithm was retrained an average of 1.0 times (median= 1.0, σ = 1.0) during each of the five tasks, after the initial training. Again, editing the training data, algorithm, and features each played a part in the process of improving the models. Figure 6.5 illustrates the actions taken to build the models in the three classification tasks in B for which the model was trained more than once. Figure 6.6b summarizes the total and per-task average number of occurrences of each system interaction in B.

The cellist iteratively modified the training set, algorithm, and features in predictable ways as she worked to improve the models. To correct misclassifications occurring for a particular type of gesture, she would often add more examples of that gesture to the training set, with the proper labels. When she felt her training set was well-representative of the problem, but classification accuracy was still poor, the algorithm, its parameters, or its features were changed in an attempt to create a better model of the current data. Occasionally the algorithm parameters were changed to address very specific goals, such as increasing the value of k for k-nearest neighbor to make the model's labels more consistent through small changes in gesture.

In general, the attempted improvements to the model were effective, in that they resulted in a subjectively improved model rating following retraining. The average increase in rating from one iteration to the next was 0.48 (median= 0.5, σ = 2.1), and the average increase in model rating from the first iteration of the problem to the final iteration was 2.7 (median= 2.0, σ = 2.0).

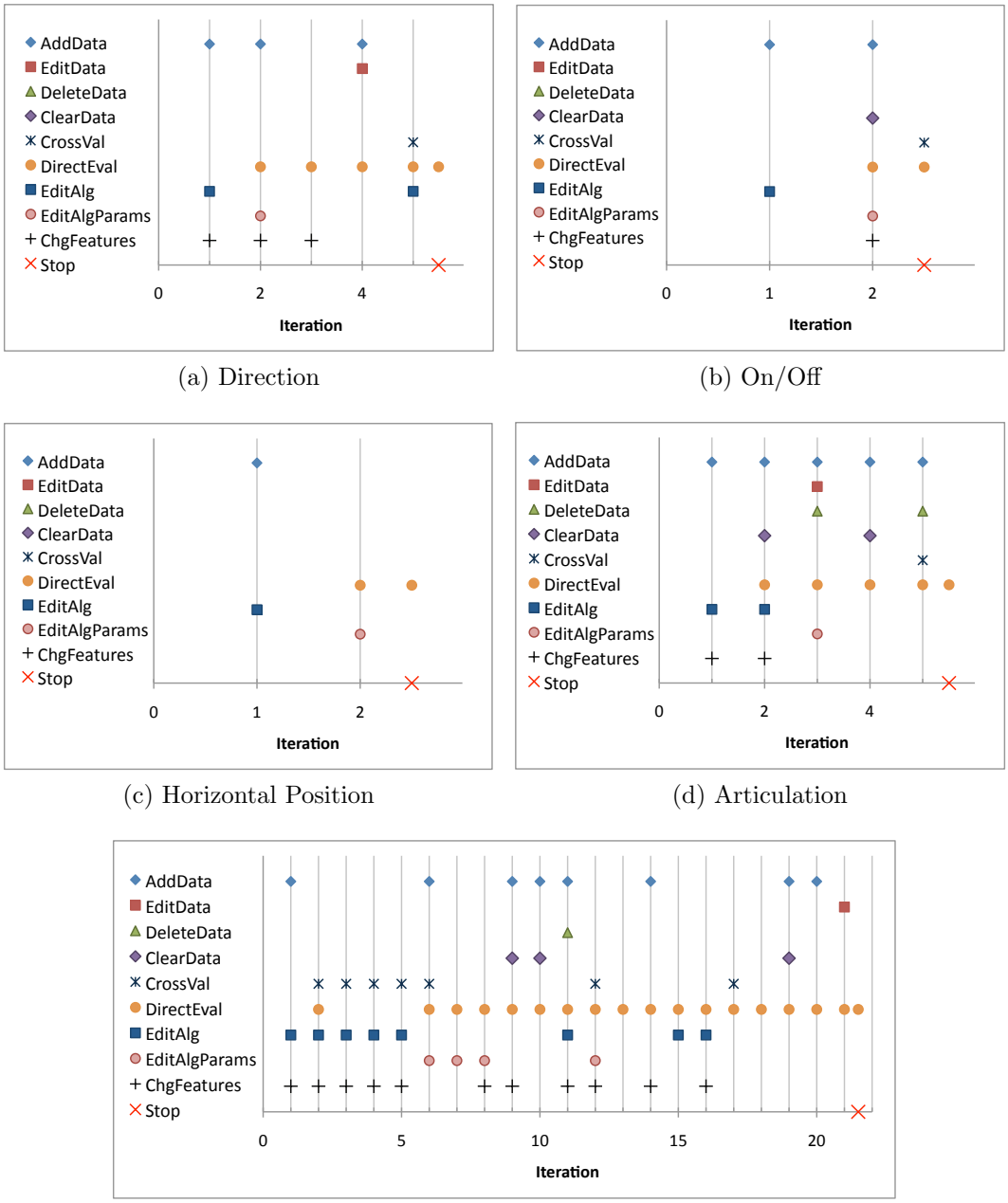
The training datasets were small enough that the training process did not interrupt interaction with the system. Figure 6.7 shows the size of the average and largest training dataset for each classification task, and the average and maximum training time per task, for all trainings in A and B. Over all, the average model training took 4.4 seconds (median= 0.2, σ = 17.7). In total, 204.9 minutes were spent interacting with the Wekinator to build the eight classifiers in A, and 44.4 minutes were spent to build the five classifiers in B.

6.5.2 Bowing Gesture Models

The best-rated models developed for each classification problem are described in Table 6.3. As the table shows, it was possible to construct models rated highly by the cellist (as "9" or "10") for all classification problems. The cross-validation accuracy computed on the training set of each of these models is also moderately high, with a minimum of 83.5% and an average of 91.8%.

6.5.3 Techniques and Criteria for Model Evaluation

Many of our recorded observations and logging concerned the ways in which the cellist evaluated trained models, the criteria she used to assess a model's quality, and



(a) Direction

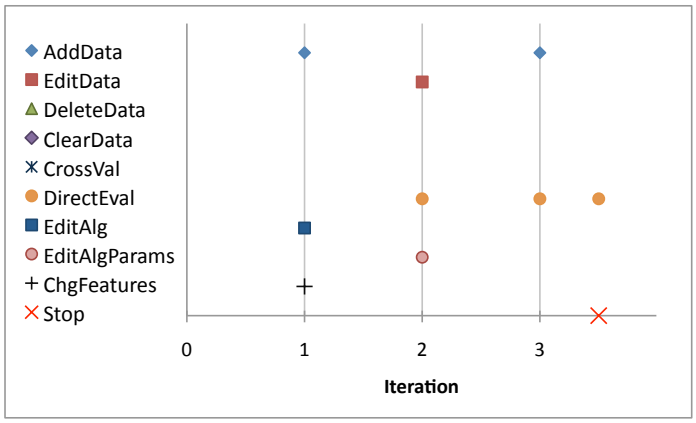
(b) On/Off

(c) Horizontal Position

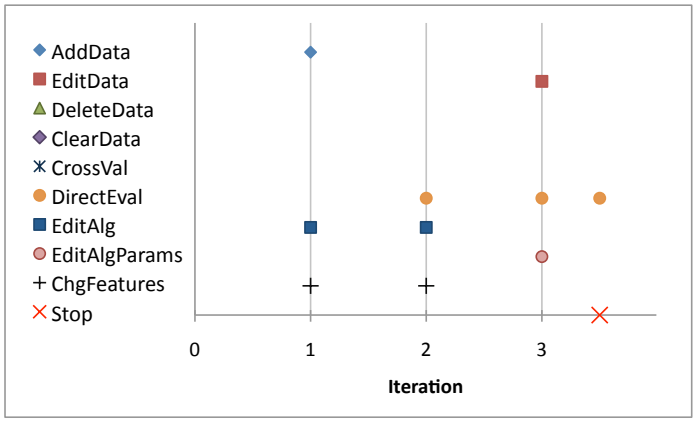
(d) Articulation

(e) Speed

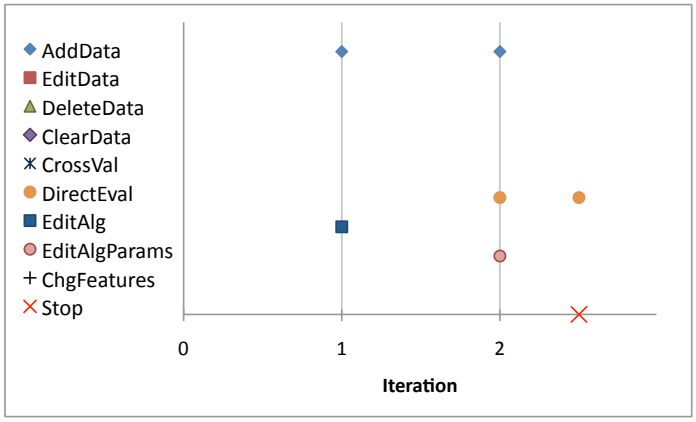
Figure 6.4: Actions taken for each task in A for which the model was retrained at least once. Each action appears above the training iteration that it precedes; e.g., between the first and second trainings of the Horizontal Position model, the model was directly evaluated and the algorithm parameters were changed. Actions above the red "Stop" × indicate actions taken after the last model training, before the cellist decided to stop building the model.



(a) On/Off

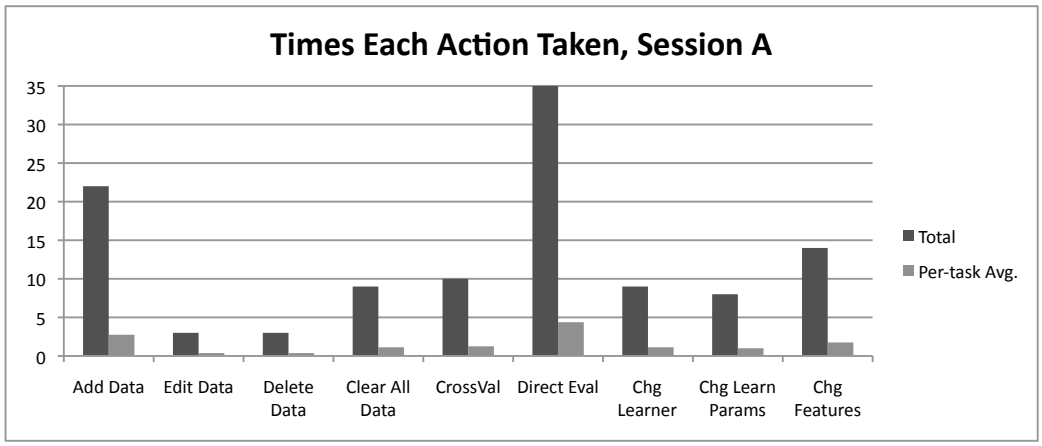


(b) Horizontal Position

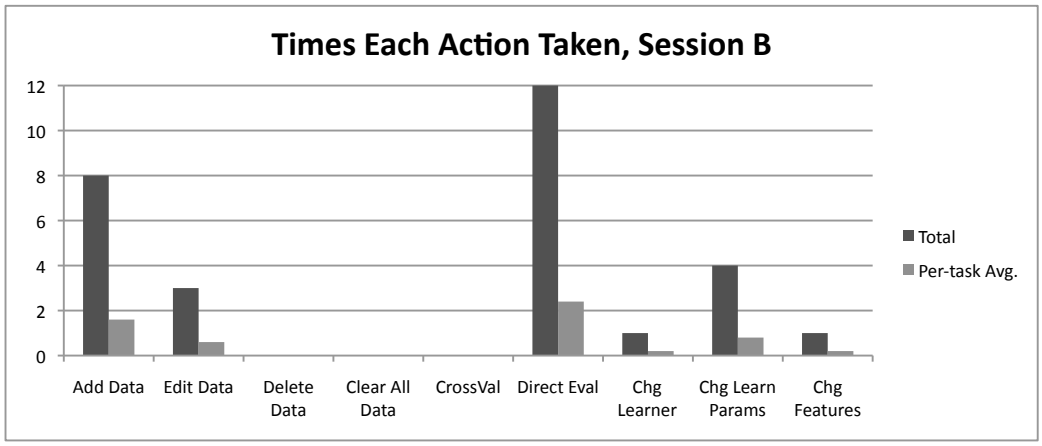


(c) Vertical Position

Figure 6.5: Actions taken for each task in B for which the model was retrained at least once. See Figure 6.4 caption for more information.



(a) Total and per-task average, in Session A.



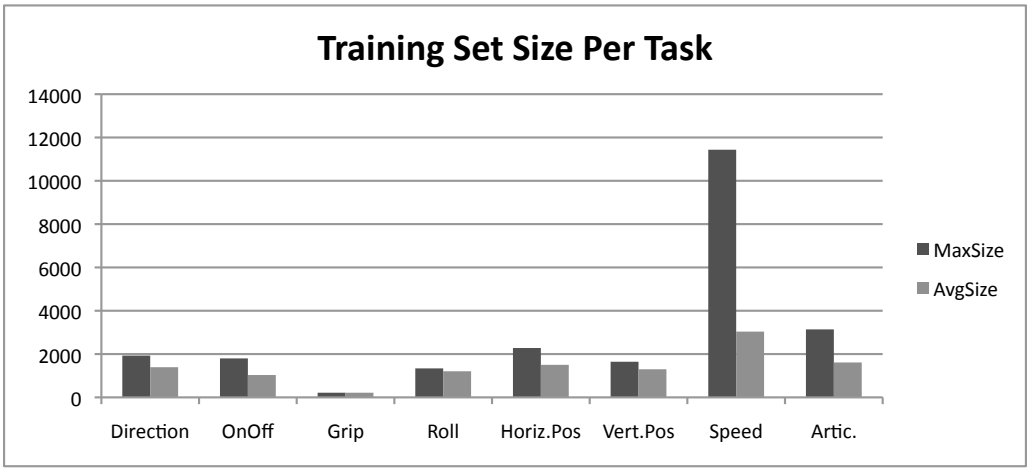
(b) Total and per-task average, Session B.

Figure 6.6: Summary of the number of times each action was performed in between retrainings of the model. Here, if an action was performed more than once between subsequent retrainings, it is only counted once.

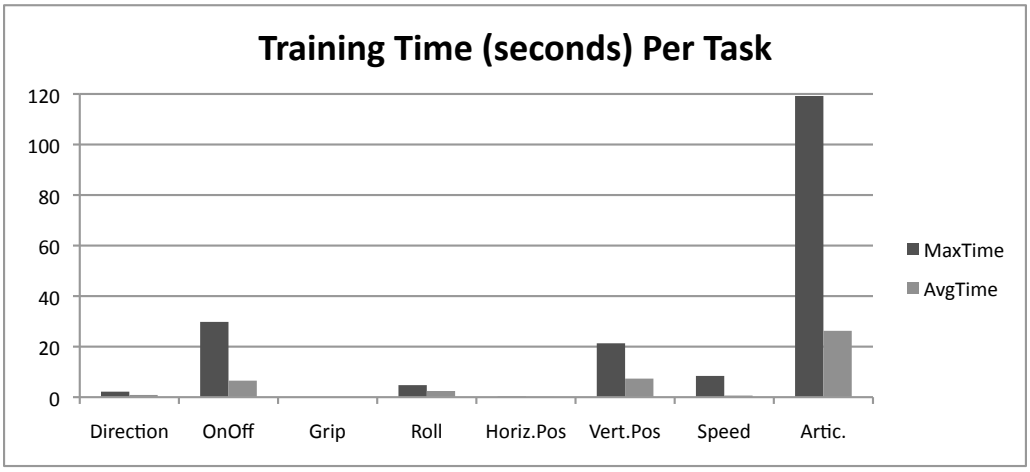
the ways her evaluations informed her interactions with the system. In this section, we discuss our observations with regard to the two evaluation techniques employed: cross-validation, and running the trained model on new gestural inputs in real time, which we will refer to as direct evaluation.

Cross-Validation

Cross-validation was used only in Session A, and it was computed 14 times in total: once for On/Off, once for Direction, once for Articulation, and 11 times for Speed. Cross-validation was used for the Speed, Articulation, and Direction tasks after direct evaluation had revealed the learning problem to be particularly stubborn, and several different algorithms and feature selections were tried in succession to see if any one might result in a usable model. Cross-validation was convenient for this purpose



(a) The maximum and average training set size, per task.



(b) The maximum and average training time, in seconds, per task.

Figure 6.7: Training set sizes and times.

because it provided a faster and more consistent way of comparing models than direct evaluation: each round of cross-validation took, on average, 1.1 seconds ($\sigma = 1.5$).

Direct Evaluation

When the cellist wanted to assess the performance of a model, to decide whether the model performed satisfactorily and she could start a new classification task, or to assess how the model might be improved, she ran the trained model as she demonstrated bowing gestures in real-time. During this direct evaluation of a model, the cellist had the option of displaying its outputs textually in the Wekinator GUI or using them to control a live visualization, as discussed below.

Because the cellist was asked to assign the current model a subjective rating from 1 to 10 based on her direct evaluation, this entailed a minimum of one direct evaluation per classification task. It took her an average of 44.1 seconds of hands-on evaluation to make a rating judgement (median = 35.5, $\sigma = 31.3$). On average, she spent 52.8

Table 6.3: The models rated by the cellist as the best for each classification task, built in the first model-building session, "A." The columns, in order, include: the task, the cellist's rating (on a 1–10 scale), the number of training iterations performed before producing this model, its 10-fold cross-validation accuracy, the number of discrete classes, the classification algorithm, and the features used by the classifier. The features are computed by the feature extraction application described in Section 6.3, computed from the base features listed in Table 6.1

Task	**Rating**	**Iter.**	**CV**	**Classes**	**Classifier**	**Features**
Direction	10	5	87.3	4	kNN	l_{vmin}, l_{vmax}, l_{vmean}, l_{amin}, l_{amax}, l_{amean};
On/Off String	10	2	83.5	2	AdaBoost on J48	h_{mean}
Grip	10	1	100.0	2	kNN	g_{mean}
Roll	10	1	98.2	2	AdaBoost on J48	min, max, and $mean$ of x, y, and z; b_{min}, b_{max}, b_{mean}
Horizontal Position	10	2	89.3	3	kNN	l_{raw}, l_{mean}
Vertical Position	10	1	90.0	3	J48	b_{raw}, b_{mean}
Speed	9	21	87.5	5	AdaBoost on J48	l_{vmin}, l_{vmax}, l_{vmean} l_{amin}, l_{amax}, l_{amean}
Articulation	9	5	98.8	7	SVM	all

seconds (median = 36.9, $\sigma = 52.2$) in each round of direct evaluation, and in total, 38.3 of 249.2 minutes with the system were spent directly evaluating the models. The cellist directly evaluated models an average of 5.4 times per task in A (median = 1.5, $\sigma = 7.6$) and an average of 2.6 times (median= 3, $\sigma = 1.7$) per task in B.

The cellist's verbal comments during direct evaluation and her choices to act to correct certain aspects of model behavior indicated that her criteria for assessing a model's quality included its correctness, subjective cost or severity of errors, decision boundary shape, and posterior probability distribution shape. We elaborate on these criteria in the next sections.

Correctness and Cost

In all bowing classification tasks except for Speed, the categories being learned by the model were defined by musical convention. (In the Speed task, the cellist herself designated certain ranges of speed as "fast," "very fast," etc.) When a model's clas-

sifications output during direct evaluation deviated from what was musically correct, this was predictably judged by the cellist to be an incorrect model behavior. This behavior was most often addressed by adding additional training examples to the training dataset, which were similar to those that caused the misclassification, but with the correct labels. In extreme cases of failure, the cellist recreated the training set from scratch. The cellist did indicate that different misclassifications had different degrees of severity, based on the musical appropriateness of the classifier's label. For example, classification mistakes a human cellist might easily make were less problematic.

Decision Boundary Shape

The cellist occasionally complained that, as she gradually changed from one bow gesture to another, a classifier's output might jump around unpredictably before stabilizing. When classifying Horizontal Position, for example, it was very important to her that the classifier cleanly switch from a label of "frog" to a label of "middle" at some point during a down-bow stroke, rather than jump between the two; it was less important that this label switching happen at a precise horizontal position. In other words, the shape and smoothness of the classifier decision boundaries in the gesture space were more important than their exact locations. Actions taken to smooth jagged decision boundaries included changing algorithm parameters (e.g., increasing k in k-nearest neighbor) and adding smoothly-labeled training data along the boundary area.

Confidence and Posterior Distribution Shape

When evaluating a model's quality, the cellist also took into account the shape of the model's estimated posterior probability distribution over the class label set. The cellist was a proficient programmer, and she worked with us during the study to design several simple visualization applications to help her understand more about the model during direct evaluation. One frequently-used visualization, shown in Figure 6.8, displayed the estimated posterior distribution as it changed in real-time, using a set of sliders. When the model classified her current bow gesture correctly but also assigned relatively high posterior probabilities to several incorrect labels, the cellist expressed dissatisfaction (at one point exclaiming, "Come on, be more sure than that!") and attempted to improve the model's confidence. In the Horizontal Position task, she actually changed her model rating from "10" to "9" when observing that the distribution appeared to gradually change between the "middle" and "tip" classes but not between the "frog" and "middle" classes while she bowed.

The cellist also considered the information gained from the posterior distribution as potentially helpful for improving the practical usefulness of a poorly-performing classifier. For example, when evaluating an Articulation classifier using the posterior visualization, she noticed that although the model often output the wrong label for three of the articulations, the posterior distribution for these articulations had a

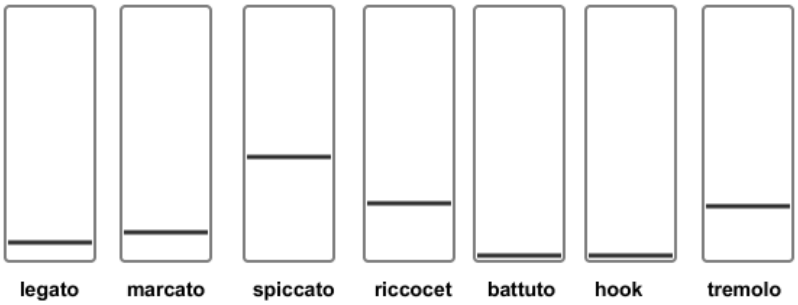


Figure 6.8: The posterior distribution visualization used for the articulation task. The position of each vertical slider indicates the classifier's estimated posterior probability that each corresponding label is correct, given the current bow features.

predictable "signature" shape that could be post-processed by some simple code to produce the correct label.

6.5.4 Human Learning and Adaptation

The cellist modified her own goals and behaviors throughout the interactive machine learning process. She remarked that her strategy for providing training data "definitely evolved over the training sessions." By the end of the study, her strategy for classification problems she knew from experience were easier to model was to provide as varied a training dataset as possible, varying "which string, bow position (frog to tip and fingerboard to bridge), [and] speed and preparation (i.e., how high off the string I would start)..." to make the trained model maximally robust to these effects. On the other hand, for problems that she discovered were more difficult to model, she started by simplifying the problem represented in the training dataset, keeping variables such as speed and choice of string constant across all training examples in order to build a model more likely to discriminate between classes based on only truly relevant criteria. In this, the cellist intentionally used the training set to represent the scope of the classification problem, just as was done in the prior bow classification research discussed in Section 6.2.1. However, here, this scoping was fluid, changing from problem to problem and over the course of a task, in response to both how the cellist anticipated using the model in performance as well as how well the model was performing.

There were no significant differences between model ratings in A compared to those in B; final model ratings for the five classification tasks completed in both A and B were identical, and a paired t-test comparing initial model ratings for each task yields $p = 0.88$. Nor was there a statistically significant difference in the number of iterations performed for each task (paired t-test yields $p = 0.55$), though the average number of iterations for these tasks was 5.4 in A and 2.6 in B, due in large part to the fact that it took 21 iterations to create a satisfactory Speed classifier in A, but only 1 iteration in B. Also, the total time spent on these tasks in B was markedly less

than in A (44.6 minutes in B versus 118.5 minutes in A), but this is due primarily to the reduced time spent on the Speed task, and the paired t p-value is not significant ($p = 0.32$). One interpretation of these results is that, while the knowledge gained in the thorough and long investigation of training data creation strategies, learning algorithms, and features undergone in building the Speed classifier in A was clearly beneficial in building the Speed classifier in B, the one or two training rounds devoted in A to each of the other four tasks (On/Off, Horizontal Position, Vertical Position, and Roll) did not provide noticeable benefits to attempting these tasks a second time.

The cellist's goals for the models, though constrained by the need to apply musically appropriate labels to natural performer gestures, were still sometimes adjusted to reflect what a model was able to learn. For example, in the Speed task in session A, after building a Speed classifier that worked well for three classes of speeds, she decided to try building a finer-grained Speed classifier for five classes. As another example, the cellist started the Bow Direction classification task with only three classes, "up-bow," "down-bow," and "not moving." However, when the initial trained models did not perform as well as expected for this very simple classification task, a fourth class was added. This class, "none of the above," was used to represent the state of the bow when it was changing direction.

Interaction with the Wekinator also led the cellist to gain a new perspective on her own bowing technique, when occasionally she discovered through consistently poor model performance that her training data was not as clear as she thought it had been. For example, noticing that the bowing articulation model was not discriminating well between riccocet and spiccato strokes, she reexamined her own technique for those strokes and discovered that her spiccato technique actually needed to be improved in order to be less like riccocet. After adjusting her technique, she was able to both train a model that performed better and produce a better cello sound. As a second example, the cellist learned through many subsequent failures to produce a working Speed classifier in Session A that her bow speed was often inconsistent and did not match her perception of how fast she was bowing. In reality, her perception of bow speed had more to do with the speed at which she was playing notes (i.e., by moving her non-bow-hand fingers on the strings) and the speed at which she was changing bow direction (which was increased by shortening the length of the bow used in each up-bow and down-bow, not increasing the horizontal bow speed). Neither of these components of bowing speed were well-captured by the bow length sensor features being used to classify speed, so the model performed poorly. The cellist decided that her goal for the Speed classifier was nevertheless to classify the horizontal speed of the bow, not her perception of speed, so she trained herself to consistently move the bow at set speeds by attaching colored markers to her bow and watching them as she played. Through this technique, she was able to create a cleaner training set and a model that she liked.

6.5.5 Cellist's Final Evaluation of the Wekinator and Models

The cellist moderately to very highly agreed with each of three statements regarding the usefulness of the Wekinator. On a 5-point Likert scale, she rated "The Wekinator

was able to create accurate bow stroke classifiers in our work so far" as a 4. She had two responses for the rating of "The Wekinator was able to create bowing classifiers that would be useful in performance": indicating that "they could be used in performance, but would have to be combined with other factors in order to make [them] truly musically relevant," she rated her agreement as 3.5, but said that her rating increased to 4.5 "if you simply want a bow stroke to trigger. . . a change." That is, as a composer, she considered the job of making models musically useful to encompass far more than creating models that performed accurately; it also encompassed composing a musically appropriate context in which the models could be used most effectively. Finally, asked to rate her agreement with the statement "The Wekinator was able to create bow stroke classifiers more easily than other approaches" on the 5-point scale, she responded "10 (so 5)," indicating that the ease of creating classifiers with the Wekinator was a key advantage of the software.

6.6 Discussion

6.6.1 Subjective Ratings and Cross-Validation

We compared the cellist's subjective model ratings against 10-fold cross-validation accuracy computed on the same models, to assess the relationship between subjective model quality and the estimated generalization accuracy computed by cross-validation. Figure 6.9 plots rating against cross-validation accuracy (computed after the study's completion) for all models created and rated in both A and B.

The cellist's rating of a model was sometimes—and sometimes strongly—correlated with the estimated generalization accuracy, but this was not the case for all tasks. For each of the six classification tasks where three or more training iterations were performed, we computed the Pearson's correlation between the cellist's rating and the cross-validation accuracy. The Horizontal Position, Vertical Position, Bow Direction, and On/Off String classification tasks had negative correlation coefficients (-0.59, -0.44, -0.74, and -0.50, respectively), while speed and articulation classification tasks (for which there were considerably more trainings performed) had positive coefficients (0.65 and 0.93). Computing the Pearson's correlation between cellist rating and cross-validation accuracy over all models and all tasks yields a coefficient of $r = 0.69$. Computing the Spearman's rho rank correlation—a measure of correlation without an assumption of a linear relationship (Wilcox 2010, 178)—yields a coefficient of $\rho = 0.68$.

The finding of a negative correlation between subjective rating and cross-validation score for some tasks seems unexpected in the context of prior work in building music classification systems. For example, in all work discussed in Section 6.2.1 above, cross-validation and test accuracy are the only evaluation methods used to assess the trained models' performance. The underlying assumption is therefore that cross-validation and test accuracy are good indicators of how well a model will perform in practice. In the four tasks here with negative correlation coefficients, models with high cross-validation accuracy and a low user rating can be explained as

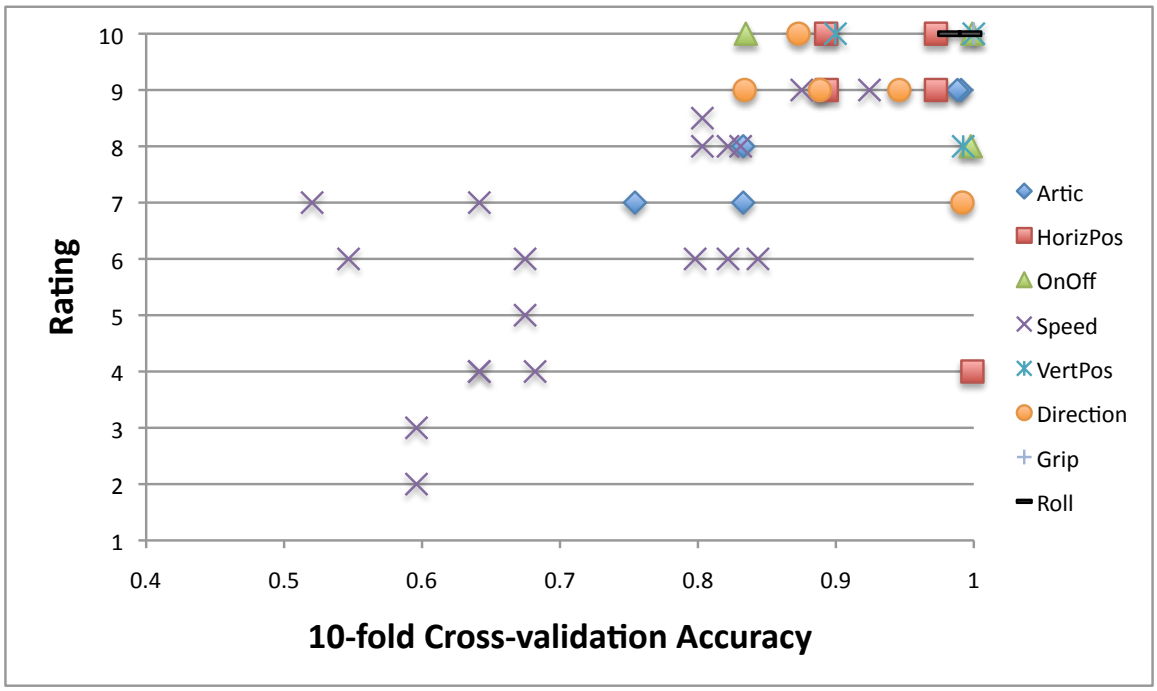


Figure 6.9: The cellist's subjective rating and the 10-fold cross-validation accuracy score for all models created in A and B, grouped by classification task.

the result of the training set providing a representation of the learning problem that was both inaccurate and too simple. For example, an initial version of the horizontal position training set contained mislabeled examples for all instances of one class. The training dataset was easy to classify correctly but the resulting model was useless. In the other tasks, this negative correlation likely resulted from the cellist unknowingly co-varying the bowing class of interest with more easily distinguishable aspects of the gesture (such as the string being played), effectively leading the model to learn the wrong concept. In all four cases, problems with the training set were undetectable using cross-validation, whereas direct evaluation allowed the cellist to discover the problems, fix them, and ultimately create models rated "10" for each task.

The sometimes negative correlation between cross-validation accuracy and subjective rating suggests that the training set may be a poor resource for estimating generalization performance during certain stages of the interactive model creation process, especially when the user has not yet discovered problems with the training data. We further discuss implications of this finding in Section 8.5.2.

6.6.2 Efficacy of Interactive Supervised Learning with the Wekinator

The high quality ratings assigned to the final models in Table 6.2 and the cellist's overall assessment of model accuracy and usefulness presented in Section 6.5.5 indi-

cate that these models are of a quality high enough to be used in performance. (The cellist/composer is currently working to integrate them into new compositions.) Because our work only concerned a single cellist, we do not claim that our classifier suite will generalize well to the bowing of other performers. Through future work with a larger pool of string players, we could apply this process to creating a more robust set of classifiers for wider distribution. However, a more immediate future application of this work is to produce a K-Bow-specific version of the Wekinator, with algorithms and features pre-selected for standard bowing gestures, and with an interface for K-Bow users to interactively provide and refine their own training examples.

The final Articulation model's cross-validation score of 98.8% is better than or comparable to previously reported results for bow classification by Rasamimanana et al. (2005), Peiper et al. (2003), and Young (2008), even while it is capable of discriminating among more articulation classes than those studies. While there is still room to improve the performance of the Articulation model, and while cross-validation is a problematic metric for reasons discussed above, this and other study outcomes indicate that an interactive supervised learning approach can build models that are at least as accurate as the non-interactive, though methodical, approaches to training data creation and model design described in the literature. In contrast to a conventional machine learning approach, the interactive process enabled the user to effectively improve the models' performance on each task through a variety of strategies, including modification of the training data. Additionally, the interactive direct evaluation process enabled the user to identify problems with a model at a fine level of granularity, as she explored the models' outputs for specific gestural inputs. This exploration allowed her to to judge a model's quality according her expert knowledge of how it should behave for particular gestures, which in turn allowed her to drive improvements to the model based on precise knowledge of how she wanted the model's performance to change. Interaction thus offers a more direct way of building models based on users' priorities, compared to the more coarse-grained strategy for model improvement used in a conventional supervised learning approach, which is to search for algorithms and features that yield higher overall accuracy. Furthermore, the interactive process enabled the user to make adjustments to the problem definition and scope, and to her strategies for providing the most effective training data, based on her evolving experience with the system.

6.6.3 The Wekinator Software

Building the bow gesture classification models engaged the use of nearly all of the Wekinator's current interfaces and features. The spreadsheet dataset editor was used for deleting segments of training data that were noisy or contained mistakes, and the graphical dataset editor was used extensively for visualizing the sensor features and manually cleaning and re-labeling the training data. Both cross-validation and direct evaluation were useful mechanisms in the interactive model building process: cross-validation allowed an efficient means of comparing algorithms on the same data, and direct evaluation using a variety of visualizations of the model outputs allowed the cellist to explore models' behavior and evaluate them against her own subjective

criteria. The foot pedal input was critical to allowing an uninterrupted, hands-free means of training data collection and model evaluation.

Several additional features could make the Wekinator a better tool for this sort of gestural recognition task, to allow for more flexible handling of features and support more appropriate modeling algorithms. First, for complex modeling tasks such as articulation, it was not clear a priori which features that could be computed from the bow sensors would be most useful for classification. Based on previous published work finding that minima and maxima of acceleration and velocity were highly relevant features, we integrated these measurements into our feature extractor application. However, similar gestural modeling problems in music—for example, identifying the beat patterns of a conductor—would likely require a different set of features to be computed from the sensor or video input features. (Notably, analysis of human conducting and interfaces for musical control using conducting gestures have been studied extensively, for example by Boie et al. 1989, Lee et al. 1992, and Nakra 2000, among many others.)

Experimentation with different types of features is likely to be necessary for complex modeling problems for which the user is not able to redefine or restructure the problem so that it can be modeled appropriately with the given features and algorithms. The Wekinator itself currently offers no way to experimentally add new features using its GUI, other than those already offered in the set of standard meta-features. Support for interactively designing new features to be computed from the existing feature set, and for retroactively adding them to the current training set, could be very helpful for these types of problems.

Additionally, much of the prior work on bow gesture recognition has applied various segmentation methods to the incoming sensor values, so that a new data example is computed for each note, as in work by Rasamimanana et al. (2005), or each full bow, as in work by Peiper et al. (2003). The Wekinator does not currently support any such segmentation; rather, each incoming feature vector is treated as a separate instance to be added to the training set or classified. This can make the learning problem more difficult, in that it can lead to a higher degree of variance among training examples within a given class; for example, the "spiccato" training instances were extracted not only from moments in time when spiccato notes were being played, but also from moments in time where no articulation label truly applied, such as moments in between notes and during changes in bow direction. Segmentation methods are useful in other classification problems beyond gesture analysis, as well; in music information retrieval, for example, a common method of extracting feature vectors from a music signal is to first detect the locations of note onsets, then extract a single feature vector per note (West and Cox 2005). Therefore, it may be sensible to add support within the Wekinator for specifying a segmentation mechanism that will dynamically identify moments in time when a training or classification instance should be constructed, and accordingly readjust the windows over which meta-features are computed.

Finally, though the current suite of classification algorithms performed well on these gesture classification problems, there are other bow gesture analysis tasks for which other learning algorithms might be more appropriate. For example, the cellist

was also interested in building classifiers for user-defined control gestures, such as the discrimination among digits (e.g., "1" through "5") drawn in the air with the bow. In a brief experimentation with digit classification, we were able to construct a 5-digit classifier that worked reasonably well (rated "8" by the cellist and obtaining 98.9% cross-validation accuracy), but the performance of the model was quite sensitive to the speed of the gesture and the size of the "history" meta-features used. While this type of task can be supported moderately well using the "history" meta-feature, which represents temporal variation directly in the feature vector by concatenating a set of the most recent values of an incoming feature, other learning algorithms that are structured to take temporal behavior into account might be more appropriate, such as hidden Markov models (Bishop 2007).

Additionally, the Wekinator's current set of algorithms cannot meaningfully use training examples that have partial or ambiguous class memberships. This type of designation would be appropriate, for example, when switching between up and down bows, or even changing from one articulation to another. A learning algorithm capable of effectively handling these designations could make the training example creation process more natural (e.g., the user would not have to create artificial additional classes for ambiguous examples, as was done in the Bow Direction task above), and it might produce more meaningful posterior distributions (e.g., representing a smoothly changing decision boundary as a gesture moves from one class to another). Therefore, future work might also examine incorporating heuristic methods for handling partial class memberships into the Wekinator's existing set of algorithms, and/or adding probabilistic learning algorithms that can handle ambiguity in a principled manner.

6.6.4 Further Discussion

We further discuss the implications of this work regarding model evaluation, algorithm and interface design, and the larger role of interaction in supervised learning in Chapter 8.

6.7 Conclusions and Future Work

In this chapter, we have discussed the application of the Wekinator to classification of standard cello bow gestures. Through an interactive machine learning process, a cellist was able to construct working models for eight bowing gestures. The cellist's subjective rating of these models was quite high—"9" to "10" on a 10-point scale—and their cross-validation accuracy is comparable to or better than previously published results in bow gesture modeling.

The most significant findings of this work entail a greater understanding of the range of interactions with the training data, algorithms, and features that support building an effective classifier for natural musical gestures; an understanding of the priorities and evaluation criteria of the cellist and information about how the subjective quality or usefulness of a model might not be well-described by a cross-validation accuracy score; and confirmation that an interactive supervised learning process can

still lead to and benefit from evolutions in users' knowledge and goals, even for modeling problems whose structure and goals are fixed and beyond the user's control.

We have discussed above the future improvements to the Wekinator software that could be helpful in modeling similar problems. Additionally, a wider study with more cellists (or other string players using the K-Bow) would be valuable in gaining a broader perspective on inter-player variations in bowing techniques and in studying other musicians' criteria for evaluating model correctness, their strategies for effectively creating models, and the ways in which their interactions with the system inform them about machine learning and about their own technique. While some prior work in bow gesture classification has motivated by pedagogical applications, the motivation seems to focus on the use of models trained by experts to give feedback to novice players. In contrast, a new vein of work might also use the process of interactive model building itself to elicit critical reflection from students as they are prompted by the system to consider how to provide a set of gesture examples that successfully demonstrate the essential characteristics of a particular technique.

Chapter 7

Case Studies: Compositions Completed by Wekinator Users

7.1 Introduction

In this chapter, we discuss case studies of three composers—a faculty member, a graduate student, and an undergraduate student—who have employed the Wekinator in the composition and performance of new musical works. In each of the following sections, we draw on informal interviews with the composers to illustrate how the Wekinator was used in their compositions, present their reflections on how the software was most useful to their work, and discuss possible improvements to the software that are suggested by their experiences. Following our discussion of the three compositions, we present a brief summary of how composers used the Wekinator, which aspects of the software were most useful to them, and how it influenced the compositions they created. We further draw on the work presented in this chapter in our discussions of interaction in supervised learning in Chapter 8 and of the use of interactive supervised learning in creative contexts in Chapter 9.

Our own role in the compositions described below was minimal. We seldom discussed the compositions with composers as they worked, and on a few occasions, we implemented new features or fixed bugs in response to users' requests. The compositions presented here constitute users' own work, and they illustrate the types of projects that end users of the Wekinator may accomplish on their own, using the version of the software that was presented in Chapter 3.

Overall, these users found the Wekinator to be a tremendously useful tool that enabled them to approach their work in new and more effective ways. Each composer employed an interactive approach to successfully building supervised learning models for use in their instruments and compositions, and they used interaction with the Wekinator to find inspiration, to take advantage of new ideas, and to work around challenges they encountered. The composers valued that the Wekinator allowed them to prioritize expression and embodiment, both during the compositional process and

Figure 7.1: A performance of *CMMV* by PLOrk in April 2010. Drum Machinists sit in the center of the stage, surrounded by Silicon/Carbonists, Tethered-uBlotarists, and guest percussionists Cameron Britt and Anders Åstrand.

in the compositions they created. They expressed that the Wekinator enabled them to approach the process of composition in a new way and to create new and more expressive types of instruments.

7.2 *Clapping Machine Music Variations* by Dan Trueman

Daniel Trueman is a faculty member in Music Composition at Princeton University. He is an active composer and performer of music for laptop, the Norwegian Hardanger fiddle, and a variety of new and folk music ensembles. Trueman was a co-founder of the Princeton Laptop Orchestra, and he has created, composed for, and performed with several original computer music interfaces, including the BoSSA (Trueman and Cook 2000). His work at Princeton has also included teaching, advising, and research activities focused on the development of new instruments and interfaces for computer music performance.

Trueman employed the Wekinator in the composition and performance of the piece *Clapping Machine Music Variations*, or *CMMV*, which is written for a variable-sized chamber ensemble of acoustic and laptop musicians. The piece was performed at Princeton by members of PLOrk on 3 April 2010, at the International Computer Music Conference by the Sideband ensemble on 5 June 2010, and by participants in the So Percussion Summer Institute on 25 July 2010. Figure 7.1 shows PLOrk students performing the piece at the April concert. Audio and video from the PLOrk and So Percussion Institute performances may be viewed online at `http://www.cs.princeton.edu/~fiebrink/thesis/resources.html`.

Trueman's program notes (Trueman 2010b) for the piece read: "At the core of *Clapping Machine Music Variations* is a pair of laptop-based Drum Machinists. Sur-

rounding this duo is an assortment of other instruments, some clearly defined laptop-based instruments, others more variable and traditional in type. *CMMV* takes specific inspiration from works by Steve Reich, György Ligeti and Björk. In particular, the drum-machine algorithm was initially designed to mimic certain rhythmic processes in the Ligeti *Études pour Piano*, processes which also coincidentally generate the rhythmic pattern for Reich's *Clapping Music* (this should come as no surprise, as both composers were deeply influenced by traditional African rhythms); this algorithm is then used to generate variations on the original *Clapping Music* pattern, variations that are explored over the course of *CMMV*. More generally inspiring are pieces like Riley's *In C*, and Andriessen's *Worker's Union*, where some things are specified, other things are not, and anyone can join the party."

CMMV is written for three sets of human performers: "Drum Machinists,", "Silicon/Carbonists," and "Any Instrumentalists." The two Drum Machinists employ a laptop GUI and a MIDI keyboard to control the delay-line-driven rhythmic patterns and drum sample instrumentation of a ChucK drum machine. The Silicon/Carbonists manipulate timbral and dynamic properties of sound produced by a granular sampling synthesis patch, also written in ChucK. The piece requires an even number of at least two Silicon/Carbonists. The Any Instrumentalists can play any instrument, and they are also matched in pairs.

The Wekinator was used in the creation of the instruments played by the Silicon/Carbonists during two of the performances, and of the "Tethered-uBlotar" instruments played by the Any Instrumentalists in all three performances to date. In the PLOrk performance, each Silicon/Carbonist employed the 3Dconnexion SpaceNavigator interface, pictured in Figure 4.2 on page 93, to drive six Wekinator neural networks controlling continuous parameters of the synthesis patch. These networks were trained by Trueman. In the ICMC performance, two performers familiar with the Wekinator (Rebecca Fiebrink and Jeff Snyder) used the Wekinator to train a new set of neural networks for controlling the same synthesis parameters using Snyder's Manta interface, pictured in Figure 7.2. Trueman modified the Silicon/Carbon mapping again in the So Percussion Summer Institute performance, this time creating an explicit mapping between a Wacom tablet and the synthesis parameters.

The "Tethered-uBlotar" instrument was created by Trueman to be capable of controlling eleven synthesis parameters of the uBlotar algorithm using the GameTrak Real World Golf "tether" controller, pictured in Figure 4.1 on page 92. This USB controller, which was used frequently by composers in the participatory design process in Chapter 4, contains two strings that a performer can pull out of the base. The controller measures the x-, y-, and z-coordinates of the two string handles in 3D space, and the Wekinator's built-in HID feature extractor (Section 3.3.2 on page 47) is able to use these six features as input into its supervised learning models. The uBlotar synthesis algorithm, described in Stiefel et al. (2004), employs a physical model capable of producing flute-like and electric guitar-like sounds, and it was used by composers in the participatory design process in Chapter 4. Here, Trueman used the Wekinator to train a set of neural networks to control eleven continuous uBlotar parameters affecting the sound's volume, timbre, vibrato, feedback, sustain, and distortion. Figure 7.3 shows a performer playing the Tethered-uBlotar in the

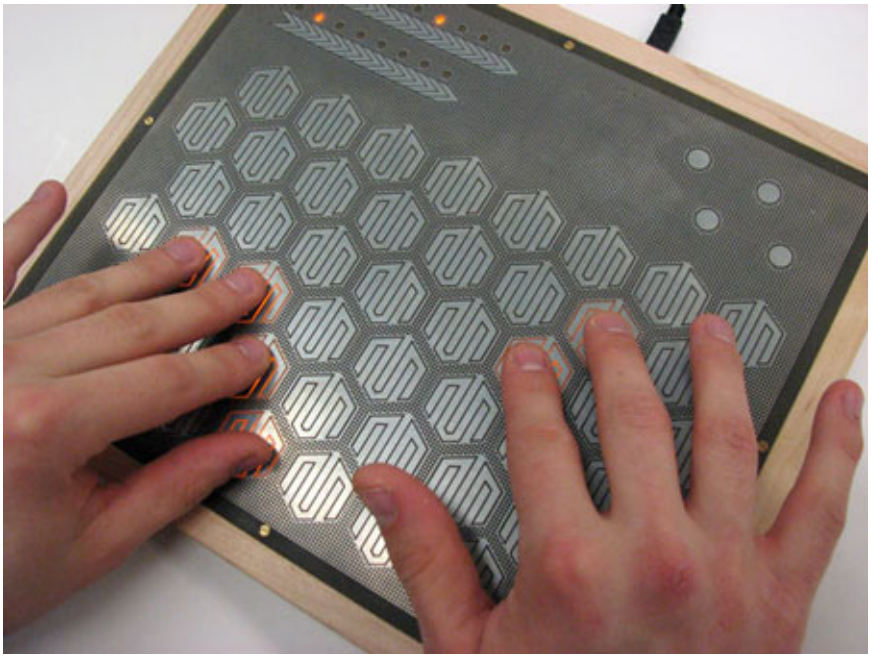


Figure 7.2: The Manta touch controller, created by Jeff Snyder 2010 and used in the second performance of *CMMV*. Each hexagon senses the surface area under a performer's touch, and programmable LEDs beneath the surface provide feedback to the performer.

PLOrk performance of *CMMV*. As the figure shows, the performance gestures for this instrument were capable of being quite dramatic and large in scale.

Our discussion of *CMMV* draws on e-mail correspondence with Trueman in October 2010, as well as on his published paper describing the piece and his compositional process (Trueman 2010a).

7.2.1 Composing the Piece with the Wekinator

Background

Trueman was actively involved in the participatory design process described in Chapter 4. During that process, he experimented extensively with the Wekinator, and he built many models using the SpaceNavigator and tether controllers, and using variants of the Silicon/Carbon and uBlotar synthesis patches. His experiences during that process informed both many changes to the Wekinator software and the development of the instruments he created for *CMMV*.

Trueman's interests include creating new sensor-based musical instruments, repurposing commercial hardware and built-in laptop capabilities as control interfaces for performing computer music, and composing and performing pieces for these new instruments. He has been building and performing with new musical controllers since the mid-1990s, and he has significant expertise in using ChucK and Max/MSP to create mappings for these instruments and write software for many other aspects of

Figure 7.3: A PLOrk students plays the Tethered-uBlotar in the first performance of *CMMV*. The performer has pulled the red strings out of the base of the GameTrak USB controller, and he is controlling the uBlotar synthesis algorithm by manipulating their positions in space.

his compositions. Prior to beginning work on *CMMV*, Trueman characterized his familiarity with machine learning as a "2" and his familiarity with the Wekinator as a "4" on a scale from 1 ("Not at all familiar") to 5 ("Extremely familiar").

Asked about the motivation for using the Wekinator in *CMMV*, Trueman wrote, "I had two instruments in mind for the piece that posed problems for explicit mapping approaches (both had many control features and many synthesis parameters) and lent themselves nicely to the implicit [i.e., generative, not explicit mapping] approach of the Wekinator."

Using the Wekinator

Trueman characterized his use of the Wekinator as initially focused on experimentally building many models to find out what he liked, then ultimately focusing more on refining a few models over a longer period of time. He trained the Tether-uBlotar and SpaceNavigator Silicon/Carbon models late in the composition process, and he distributed final versions of the trained models to performers at the beginning of the rehearsal process. He did not change the models after rehearsals began. He did, however, continue to experiment with alternative controllers for the Silicon/Carbon instrument through the subsequent performances of the piece, finding that he was unsatisfied with the SpaceNavigator instrument.

Trueman used the Wekinator to craft expressive instruments that sonically fit into the overall piece, that lent themselves to playing using subtle gestures, and that provided challenges to the performer while maintaining a predictability of control: "For this piece, I wanted an expressive instrument within a narrow sound-range. I wanted to be able to make small, subtle moves, and have them reflected appropriately in the sound. I also wanted to have, at least in part, continuity, so that most of the time there wouldn't be sudden changes in the sound world. However, I did find that I really liked having 'tipping points,' gestural areas where the model changed suddenly; these continue to be quite fun and interesting to negotiate. However, if there are too many of these, and not enough continuity, the instrument begins to feel random, and I prefer to have consistency and reproducibility in behavior."

In performances of *CMMV*, the performers launched the software component of their instruments, which contained Trueman's trained models, by running a shell script. This script launched the Wekinator using the command line options described in Section 3.5.1 on page 78 to automatically load the Feature Configuration and trained Learning System, begin running the Wekinator to produce output synthesis parameters from the extracted gestural features, and minimize its GUI. Performers therefore did not interact at all with the Wekinator GUI.

Interactions During Model Creation

Trueman used playalong recording almost exclusively to create the training data for the models, and he developed his own strategy for using the Wekinator to discover the sounds that he wanted to use in the piece and to sculpt the behavior of the trained models. He writes: "I had the most success with playalong situations, where I refined a playalong score to outline the sonic extremes of the instrument I was interested in playing, and where I then practiced playing along before creating a data set." Trueman often deleted all the training data and recorded new training examples from scratch. As he played with the models and discovered more about the sounds he wanted to use, he performed subsequent iterations of of playalong recording using modified parameter values in order to narrow the instrument's initially broad sonic range down to a "more specific sound-world." Additionally, he sometimes used subsequent playalong iterations to provide better gestures in conjunction with the same parameter sets, "so I can playalong again, but playalong *better* than I had previously; I find I need to practice!"

He found that adding more data to existing training sets was an effective way to "refine or bring out a particular feature of the sound world that [he was] after," but this action also sometimes changed the models in other ways that he liked: "The most memorable example is when I wanted to teach it to be quiet when the tethers were fully released. I added more training data to that effect, which worked well, but it also changed the way the model behaved when the tethers [were] extended, and in quite exciting ways. This was a nice, unintended result."

Trueman also experimented with adding meta-features to the models, though he states, "I feel like I need a lot more experience with the meta-features before I really

understand them and can use them effectively." He never edited the training data or deleted subsets of the training data.

Trueman agrees that he became better at using the Wekinator as he composed, "though I still feel like a beginner. I mostly got better at anticipating what kind of training sets would work well and give me something that I found interesting to play." He indicates that he developed the strategy of using playalong recording, described above, to become best able to create those types of training sets.

7.2.2 The Wekinator's Influence on the Composition

The Wekinator software influenced the composition of *CMMV* by enabling Trueman to discover new mappings and compositional ideas, challenging him to reflect on the nature and role of the Silicon/Carbon instrument when he could not create a suitable mapping for it, and enforcing particular constraints on the type of mappings he was able to easily create using playalong recording.

Table 7.1 shows the extent to which he had initial plans for the gestures, sounds, and mappings, and the extent to which he changed his mind about those aspects of the instruments as he worked. Although he began the process of composition with some clear ideas about the interfaces and sounds he wanted to use, the final realization of the instruments was continually informed by his hands-on experimentation with the Wekinator: "After creating an initial model, I spent a fair [amount] of time exploring the model, learning what was possible and interesting. I created new models and revised models many times, constantly re-evaluating based on how the instrument responded. The final instrument is not one I would have, or could have, envisaged before beginning." He also writes that the instruments created with the Wekinator could "themselves inspire new compositional ideas."

Trueman was satisfied with the Tether-uBlotar instrument that he created with the Wekinator, and he has not changed it since the first performance of *CMMV*. However, over the course of the three performances of the piece, he has experimented with a number of different physical controllers for the Silicon/Carbon instrument. Ultimately, he found that he was not satisfied with any of the mappings he created for Silicon/Carbon using the Wekinator, and the failure of the Wekinator to produce a satisfactory instrument led him to reflect on the nature of the instrument itself and to reconsider its role in the piece. He writes, "in the end I found that an explicit approach to creating a mapping for the *CMMV* granulator instrument (with [the] Wacom tablet) worked the best; somehow, the synthesis parameters suggested one-to-one mappings quite naturally, and the non-linearity of the Wekinator models were a bit frustrating. That said, in the end I think that that instrument is a bit of loser for *CMMV*, and I'm probably going to cut it completely; it may be that the fact that it wanted an explicit mapping was a sign that it wasn't a very good instrument."

Although the playalong recording process was integral to Trueman's strategy for building his instruments, he was frustrated by the way that it privileged certain types of playalong gestures and training datasets that had characteristics he felt were contrary to his compositional goals. As discussed in Section 3.4.6 on page 74, the parameter clipboard can be used as a playalong score that sequentially plays each

Table 7.1: Rated level of agreement with statements about compositional goals before and during work with the Wekinator in *CMMV*, rated on a 5-point Likert scale from 1 = "Strongly disagree" to 5 = "Strongly agree."

Statement	**Agreement**
Before I started working with the Wekinator...	
I had a specific gestural interface in mind that I wanted to use or build for the piece	5
I had specific physical gestures I wanted to use in the piece	2
I had a specific palette of sounds I wanted to use in the piece	4
I had specific ideas about how I wanted gestures to control sounds in the piece	3
While working with the Wekinator...	
I changed my mind or had new ideas about the gestural interface(s) that I wanted to use, or that I wanted to build	3
I changed my mind or had new ideas about the physical gestures I wanted to use in the piece	5
I changed my mind or had new ideas about the palette of sounds I wanted to use in the piece	4
I changed my mind or had new ideas about how I wanted gestures to control sounds in the piece	5

set of parameters for a set duration of time. It does not currently support gradual transitions from one set of parameters to the next; though Trueman requested this at the culmination of the participatory design process, it has not yet been implemented. Reflecting on this problem, he writes, "It's hard to make playalong scores that don't feel 'edgy' and 'pointy' with the sudden changes, and so it's hard to make an instrument where you want to move smoothly but the playalong is so jerky." Because it was important to him to create instruments capable of subtle and smooth changes, and to be able to practice along with the playalong scores in a natural manner, he sometimes ended up creating new clipboard-based playalong scores to work around this problem: "...with the tether-blotar, I made a fairly narrow playalong score—the parameter changes were minimal—so the pointy-ness was minimized and the playalong more closely approximated what I wanted to ultimately do with the instrument after training."

7.2.3 Evaluation of the Wekinator

As illustrated in Table 7.1, Trueman found the Wekinator to be extremely valuable as a compositional tool. He strongly agreed that it allowed him to create mappings more easily, to create more expressive mappings, to create new kinds of music, and to

Table 7.2: Rated level of agreement with statements about the Wekinator's usefulness, in *CMMV*; rated on a 5-point Likert scale from 1 = "Strongly disagree" to 5 = "Strongly agree."

Statement	**Agreement**
The Wekinator allowed me to create mappings between gesture and sound more easily than other techniques.	5
The Wekinator allowed me to create mappings between gesture and sound that were more musically expressive than other techniques	5
The Wekinator allowed me to create a kind of music that isn't possible or that is hard to create using other techniques	5
Using the Wekinator allowed me to approach the process of composition in a new way	5

approach composition in a different—and more enjoyable—way. Addressing this last characteristic, he emphasizes the usefulness of the Wekinator for rapid experimentation and inspiration during the composition process: "I like to enjoy the process of composing, however difficult it may be. I have a variety of tools and approaches towards composing that I switch between, depending on the needs of the piece and where I am creatively. The Wekinator suits this approach nicely, allowing me to create new tools quickly and on the fly. These tools tend to have personality, which suggest different kinds of music; I have always found musical instruments tremendously inspiring compositionally, and the Wekinator enables the rapid and fun construction of new instruments that can themselves inspire new compositional ideas."

Discussing the aspects of the Wekinator that were most useful to him, Trueman highlights its support for creating instruments with a broad capacity for expressiveness, that were customized to a particular composition, and that were both accessible and musically engaging to the performer: "The Wekinator enabled me to create a physically expressive instrument that was precisely tailored to the sound-world of the piece. In particular, I was able to prescribe a narrow pitch range around the primary pitches of the piece that the tether performer could subtly navigate, creating a range of timbres that are simply impossible with acoustic instruments. In performance, the physical aspects of the tether controller combine, through the Wekinator mapping, with the blotar physical model to create an instrument that is both compelling to play and watch. It requires practice to explore and master, and is constantly revealing new possibilities, but is fairly easy to get started on; the player doesn't need to spend lots of hours learning the instrument before joining the piece—rather the instrument teaches the player how to play it, especially when within the complete sound-world of the piece."

Trueman further writes that the Wekinator "enables certain kinds of instruments that simply wouldn't exist otherwise," and "I don't think I would have attempted to create this instrument without the Wekinator, given the complexity of the mapping problem, and the piece would have turned out differently." Elaborating on the type of

instruments that may be created with the Wekinator, he emphasizes both the ease of creating complex mappings and enabling the composer to privilege the physical "feel" of the instrument during the design process: "Without [the Wekinator], it's either impossible or practically impossible (just too hard and tedious to do by hand) to create certain kinds of instruments, so they never come into existence... But, maybe there is another way of thinking about it. With [the Wekinator], it's possible to create physical sound spaces where the connections between body and sound are the driving force behind the instrument design, and they *feel* right. It's very difficult to do this with explicit mapping for any situation greater than 2–3 features/parameters, and most of the time we want more than 2–3 features/parameters, otherwise it feels too obvious and predictable. So, it's very difficult to create instruments that feel embodied with explicit mapping strategies, while the whole approach of [the Wekinator], especially with playalong, is precisely to create instruments that feel embodied. I like to think of digital instrument building as a kind of choreography. choreographers are hands-on—they like to push, pull, hold their dancers, demonstrate how things should go, in order to get what they want, and the resistance and flow of their dancers in turn feeds back into their choreography. This is quite similar to the approach that [the Wekinator] engenders, and radically different than what explicit mapping strategies enable."

Trueman ties his extensive use of playalong recording to the importance he places on physicality, and he relates this to his experiences as a fiddler: "The score creation and practicing aspects are really important for me. It reminds me of learning to play fiddle tunes; takes a long time, and I have to really work on the feel of the bowings, beyond just getting the notes. I play along with other fiddlers a lot, live in jam sessions and with recordings, so the playalong notion is very familiar. I'm used to putting in a lot of deliberate work to get these tunes in my body, and I feel like I had the most success with [the Wekinator] when I could set up a situation where I could practice like that to a good score, and then train from that data."

7.2.4 Suggested Improvements

Asked what new features of the Wekinator might be most helpful to him in the future, Trueman reiterated his wish for parameter smoothing in the playalong parameter clipboard, to give him a greater freedom to practice smoothly changing gestures along with playalong scores, and to create training datasets that more effectively captured his intentions for the instrument.

He also expressed interest in adding interfaces within the Wekinator to support explicit mappings. Such support would enable more efficient experimentation with explicit alternatives to the neural network models; Trueman performed such experimentation when designing mappings for the Silicon/Carbon instrument, but he had to design the explicit mappings outside the Wekinator environment. Additionally, he saw this as being potentially quite helpful in applying the Wekinator to controlling instruments that used explicit mappings for some parameters and generative mappings for others.

7.3 *The Gentle Senses* by Michelle Nagai

Michelle Nagai is a graduate student in Music Composition at Princeton University. In her work, she "creates site-specific performances, compositions, installations, radio broadcasts, dances and other interactions that address the human state in relationship to its setting" (Nagai 2010a). A composer of both acoustic and digital/computer music, Nagai used the Wekinator throughout the conception, development, and performance of a new musical instrument, the MARtLET, and in a composition for the MARtLET called *The Gentle Senses*. *The Gentle Senses* was publicly performed on 27 April 2010 at Princeton University.

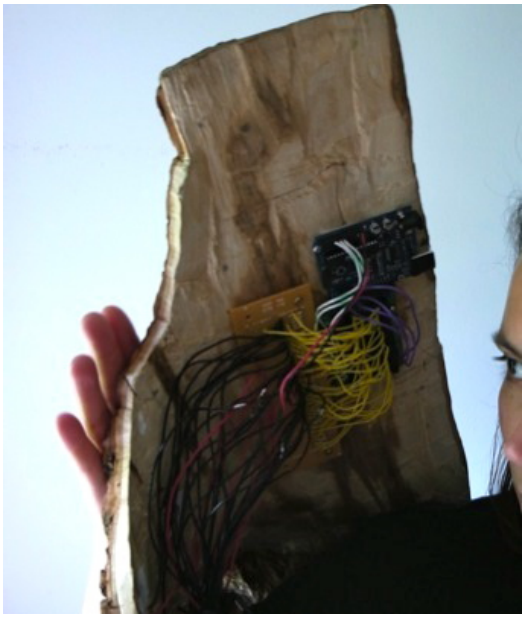
MARtLET is an acronym that stands for "Material Artifact, Responding to Light, Emitting Tones." The MARtLET interface, shown in Figure 7.4, is a wearable piece of tree bark containing 28 light sensors. Nagai designed the MARtLET to be controlled by "[t]racing inflections of light and shadow as [she moves her] hands and arms across the surface" during performance (Nagai 2010b). She describes the relationship between the MARtLET instrument and *The Gentle Senses* as follows: "The composition was developed in tandem with the construction of the instrument itself. Although I am composing new work for the MARtLET, many of the sounds and techniques that I created for *The Gentle Senses* have persisted. In this way, the first work made for the instrument has also become the voice of the instrument (in part)."

Describing her performance technique, Nagai writes, "In performance on the MARtLET, I work with very slow, subtle gestures, sometimes shifting only a few inches away or towards the light. I use my hands to seek out the possibilities for sound control across the surface of the instrument. Depending on the performance environment, I am able to get a range of sound results from this type of movement." Sonically, *The Gentle Senses* is "a quiet, slowly changing piece with a lot of delicate and subtle gradations of sound." An audio excerpt of the piece may be heard at `http://www.cs.princeton.edu/~fiebrink/thesis/resources.html`.

In the instrument and piece, the Wekinator is used in conjunction with several Max/MSP patches created by Nagai. The light sensor values are communicated via USB to a laptop, where they are first sent to a Max/MSP patch that applies filtering to the values. This patch is used by the Wekinator as an OSC feature extractor (see Section 3.3.2 on page 47), which sends the processed feature values to the Wekinator to drive four multilayer perceptron neural networks. The Wekinator communicates the outputs of these neural networks via OSC to a Max/MSP synthesis patch. In that patch, they control the register (i.e., pitch range) of the generated sounds, as well as the pulse period, amplitude, and filter Q parameters of a vocoder.

The following discussion of *The Gentle Senses* draws on an in-person interview conducted with Nagai on 27 May 2010 and on e-mail correspondence conducted in October 2010.

(a) Nagai wearing the MARtLET.



(b) The sensing hardware behind the bark.

Figure 7.4: The MARtLET instrument, created by Michelle Nagai and performed in her composition *The Gentle Senses*. Photos are by Kenta Nagai and are used with permission.

7.3.1 Composing the Piece with the Wekinator

Background

Nagai took part in the participatory design group described in Chapter 4. During that process, she experimented with using the Wekinator for building gestural control interfaces for controlling the blotar synthesis algorithm (Stiefel et al. 2004) using interfaces including the GameTrak Real World Golf controller (Figure 4.1 on page 92) and the laptop's internal motion sensor. In parallel, she worked to develop her ideas and the sensor hardware for the instrument that would become the MARtLET, and she used the Wekinator to experiment with a light sensor prototype at the very end of the participatory design process. From January 2010 until the performance in April 2010, she worked independently on the composition of her instrument and piece, using a version of the Wekinator software that incorporated the majority of the improvements resulting from the participatory design process.

Nagai had considerable prior experience working in Max/MSP before composing this piece, and she estimates that she had previously created "maybe a half dozen" pieces or instruments that involved some sort of gestural mapping before her participation in the participatory design work. She had not previously used machine

learning, however, and she rated her familiarity with both machine learning and with the Wekinator as low ("2" on a scale from 1 = "Not at all familiar" to 5 = "Extremely familiar") before beginning her work on the MARtLET.

The inspiration of the physical form of the MARtLET came to Nagai one day on a walk through Princeton, when she passed a fallen tree. She immediately knew she wanted to turn the wearable bark into an instrument, and she took it home. Nagai's plans for using the Wekinator in this new instrument took shape during her experiences with the software in the participatory design workshop sessions: "After getting familiar with the software, I began to understand the real value of the Wekinator in working with the MARtLET—namely, the possibility for gestural control that is non-explicit, changeable with each different performance environment and sensitive to subtle differences in gesture and interpretation."

Using the Wekinator

Nagai developed the trained Wekinator models in conjunction with the development of the Max/MSP synthesis patches. She describes the composition process as entailing "lots of back and forth" between adjusting how the Max/MSP patch produced sound from the parameters and retraining the Wekinator models to adjust the types of parameters it output for different gestures. She continued to refine the Wekinator's models "up until fairly late in the compositional process. At a certain point, just a few days before performance, I settled on a set of models and saved that and worked with it. I felt I needed time to learn the model before going on stage." After practicing with the final set of models, she continued to experiment with modifications to the synthesis code "right up until dress rehearsal time. I was getting [Wekinator outputs] that I liked, in terms of a steady stream of suitable numbers, and then I was switching this stream of numbers into other parameters in my algorithm, or scaling it in different ways, sort of on the fly, to see if I liked the results."

Her goal for the Wekinator was to produce models that provided her with a large expressive range while also allowing her to use a gestural vocabulary that was appropriate to the piece: "I was always searching for [models] that would provide enough sound variation, not just one or two different sounds types, but many, with interesting, unpredictable transitions in between them. I did also prefer models that responded to the most subtle movements and gestures. I didn't like models that only responded with a few big movements."

To build models with these characteristics, she developed a strategy of using the training data to map out the range of effects that she hoped to control, and to associate different areas of the instrument with the different effects: "I worked with blocking light to specific areas of the instrument and sending in different processing effects (all sound sources were sine tones, with various processing added). In this way I was trying set up zones for different textural, timbral and registral effects, hoping that I could work with them somewhat independently and also in combination with one another."

Interactions During Model Creation

Nagai primarily created training data by specifying sets of parameter values in the "Collect Data" subview of the Wekinator's "Use it!" tab (see Section 3.4.5 on page 66) and recording the corresponding gestural inputs, stating that she "never was comfortable with playalong learning. I would have liked to work more with the spreadsheet and graphical editor, but time didn't allow for that." She frequently added more data to the training set and retrained when she wanted to change a model without starting over, but she was often unsatisfied with the outcomes: "I don't think [adding data] was all that useful a strategy... It seems to just muddy what I was working with. I almost always was forced to delete and start from scratch at some point after adding more training data, when I lost my sense of where things were headed." She frequently deleted the entire training data set and started over, and she only occasionally edited the training set.

Nagai experimented extensively with using feature selection to limit which light sensors influenced each sound parameter, and to reduce the sometimes significant training time required by the neural networks. As mentioned above, she also frequently experimented with how the neural network outputs were used in the synthesis patch. She did not edit the neural network architecture or change its learning parameters.

Nagai remarked that, through her interactions with the Wekinator, she learned to use it more effectively. She also learned to take measures to reduce the training time of the neural networks; training could take up to 30 minutes, presenting an obstacle to efficient work: "I learned to limit features in my training sessions so that it would take less time to train. I also learned how to retrain only a part of model. This was very helpful. And I learned how to distinguish what part of the model wasn't working and could better focus on changing just that part."

7.3.2 The Wekinator's Influence on the Composition

As shown in Table 7.3, Nagai approached her work with the Wekinator without clear initial ideas about the gestures or sounds she wanted to use; instead, she developed these ideas during and through her work with the software. Commenting on the slow and subtle gestural vocabulary she used in the piece, she writes, "I've developed this kind of gesture, I believe, as a direct result of the responses I got from training the Wekinator. More specifically, I wasn't getting entirely predictable results from the Wekinator using gestures that were fairly consistent and specific. So instead I developed a kind of searching, sensing performance gesture that seems to work very well with the visual element of the MARtLET and allows the Wekinator to trigger sounds with more subtlety and in a more aesthetically satisfying way." Additionally, she wrote about the sound of the piece: *The Gentle Senses* turned out to be a quiet, slowly changing piece with a lot of delicate and subtle gradations of sound. I think, again, this was largely due to the kind of responses I was getting from my trainings with the Wekinator."

Table 7.3: Rated level of agreement with statements about compositional goals before and during work with the Wekinator in *The Gentle Senses*, rated on a 5-point Likert scale from 1 = "Strongly disagree" to 5 = "Strongly agree."

Statement	Agreement
Before I started working with the Wekinator...	
I had a specific gestural interface in mind that I wanted to use or build for the piece	1
I had specific physical gestures I wanted to use in the piece	1
I had a specific palette of sounds I wanted to use in the piece	1
I had specific ideas about how I wanted gestures to control sounds in the piece	1
While working with the Wekinator...	
I changed my mind or had new ideas about the gestural interface(s) that I wanted to use, or that I wanted to build	5
I changed my mind or had new ideas about the physical gestures I wanted to use in the piece	5
I changed my mind or had new ideas about the palette of sounds I wanted to use in the piece	5
I changed my mind or had new ideas about how I wanted gestures to control sounds in the piece	5

7.3.3 Evaluation of the Wekinator

As illustrated in Table 7.4, the Wekinator offered Nagai a valuable tool for creating mappings more easily, creating more expressive mappings, and creating a new kind of music and taking a new approach to composition.

Compared to the music she might have created with other techniques, Nagai valued how she was able to use the Wekinator to create music where, in her experience as a performer, "[t]here was a stronger connection between the physicality of specific gestures and the resulting sounds, more like playing a violin or some other acoustic instrument." It was important to her that this relationship was both direct and complex; because the logical, mathematical relationship between a gesture and the perceived sound was less obvious, her attention (and that of the audience members) was more focused on the experience of the piece and not on attempting an intellectual deconstruction of the mapping function.

Nagai discusses how the Wekinator enabled a new approach to composition: "I have never before been able to work with a musical interface (i.e. the MARtLET) that allowed me to really 'feel' the music as I was playing it and developing it. The Wekinator allowed me to approach composing with electronics and the computer more

Table 7.4: Rated level of agreement with statements about the Wekinator's usefulness, in *The Gentle Senses*; rated on a 5-point Likert scale from 1 = "Strongly disagree" to 5 = "Strongly agree."

Statement	**Agreement**
The Wekinator allowed me to create mappings between gesture and sound more easily than other techniques.	4
The Wekinator allowed me to create mappings between gesture and sound that were more musically expressive than other techniques	5
The Wekinator allowed me to create a kind of music that isn't possible or that is hard to create using other techniques	5
Using the Wekinator allowed me to approach the process of composition in a new way	5

in the way I might if I was writing a piece for cello, where I would actually sit down with a cello and try things out."

Asked what aspects of the Wekinator were most useful in composing the instrument and piece, Nagai responds, "... [T]he ability for sound-gesture mappings that are flexible and somewhat unpredictable (within a larger pre-determined framework). This not only contributed to the composing of the piece, but helped to influence the construction of the instrument, the gestures used for performance and subsequent revisions of both of these elements."

The most significant difficulty Nagai encountered during her use of the Wekinator was the sometimes high training time of the neural networks. Throughout the composition process, she at times used up to 15 neural networks, each with up to 28 features and several thousand training examples. In the worst case, training the entire set of neural networks took up to half an hour or more, and she found this frustrating and disruptive. Additionally, she was distressed by not having a clear idea of how long to expect the training to take, as the progress bar displayed by the Wekinator only indicates the percentage of models that have been trained, not the progress of the currently training model. (This aspect of the Wekinator's implementation is constrained by the fact that finer-grained information about training progress is not easily accessed from Weka). She did not feel like she had adequate information to choose whether to cancel the training or wait for it to complete. A significant amount of her work with the system therefore concentrated on figuring out how to reduce training time, in particular by using feature selection and only training a subset of models at any given time.

Nagai expressed a desire to edit the training data more often in the future, using the spreadsheet viewer or graphical editor. She found that deleting all the training data and starting over sometimes involved more overhead than she was willing to expend, while simply adding more training examples muddied up the models' behavior in an unsatisfying way. She saw editing the data as a viable alternative that she was more comfortable trying after gaining more experience using the Wekinator: "As I

Figure 7.5: PLOrk students performing *G*. Performers are following the Processing score on their laptop screens and playing the piece by tilting and hitting their laptops.

got more comfortable with the software, I attempted it more. In future work with the Wekinator, this will certainly be something I do more of."

7.4 *G* by Raymond Weitekamp

Raymond Weitekamp was a senior undergraduate student at Princeton in Spring 2010 when he composed the piece *G* for the Princeton Laptop Orchestra (PLOrk). Though a Chemistry major at Princeton, and now a graduate student in Chemistry, Weitekamp is an active electronic musician and performs under the name "Altitude Sickness." He describes his compositional background as "primarily rooted in sample-based music, as well as in on-the-fly looping of live instruments." Weitekamp's compositions are "heavily inspired by the hip hop and 'beat music' cultures, which are rooted in sampling," and his music frequently "recontextualiz[es] the sampled works by layering loops and non-linear patterns of those loops."

Weitekamp was a member of the PLOrk ensemble for several years, and he wrote *G* to be performed by nine members of the group during his final semester. He describes the motivation and concept of the piece as follows: "Compositionally, my motivation for *G* was to step outside of my mode of working with sample-based music; every instrument in the piece was synthesized in Chuck (in realtime). Conceptually, the original idea for the piece was to have each performer map their own physical gestures on the laptop to specific musical events. The idea was to use the play along learning algorithm at the beginning of the piece to classify which gestures (as read by the internal accelerometer) correspond to which sounds. The only controller for each of the instruments was the keyboard and [motion-sensor-driven] Wekinator output. The score for the piece was delivered in realtime using a 3-D Processing sketch, delivering notes in a 'Guitar Hero' fashion." A screenshot of this score as it was displayed on performers' laptop screens appears in Figure 7.6. Figure 7.5 shows students performing *G* during a concert on 3 April 2010 at Princeton University; a second performance was given at Princeton on 15 May 2010. Audio and video of

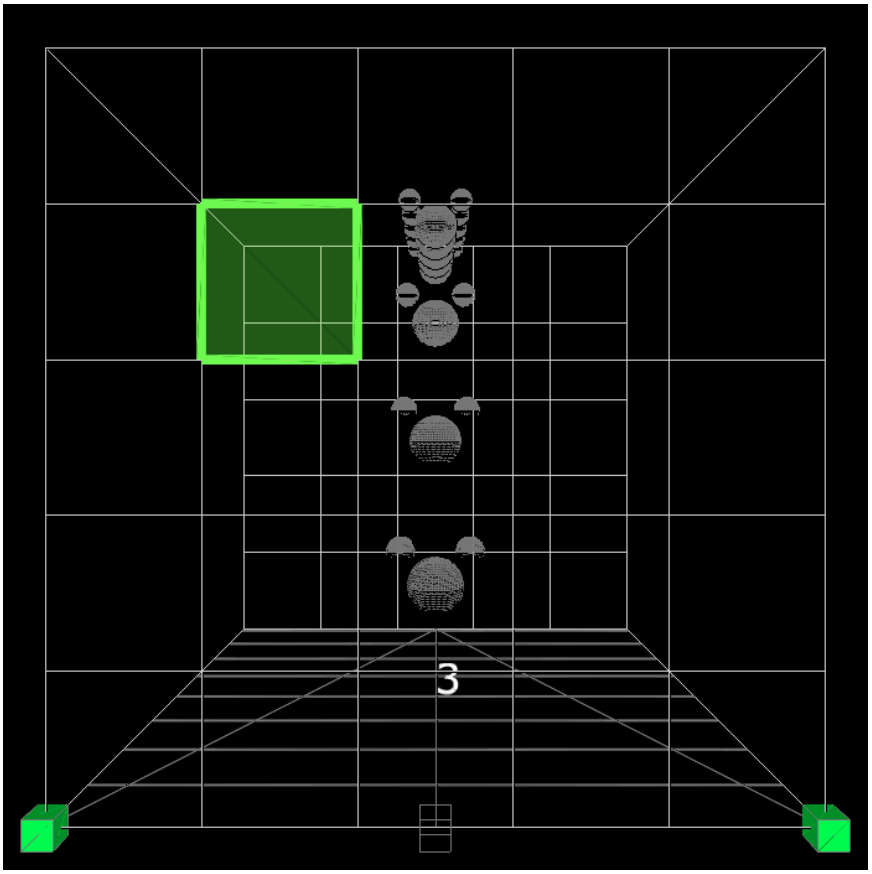


Figure 7.6: The Processing-driven score for Weitekamp's *G*. Objects in the score float through the three-dimensional grid towards the performer, and they indicate the timing and location of laptop hits as well as the timing and position of laptop tilts.

the 3 April performance can be accessed online at `http://www.cs.princeton.edu/~fiebrink/thesis/resources.html`.

Compositionally, *G* is beat-oriented and tonal, and the use of the Processing score enables a style of performance that is—compared to many PLOrk pieces—less improvisatory and more rigidly structured with regard to both the temporal evolution of the piece and the roles played by each performer. *G* employs four virtual instruments created by Weitekamp, which are characterized by distinct synthesis algorithms and distinct roles in the piece. These instruments were each played by two or three performers, and they are described in Table 7.5.

As the table shows, each instrument is played using the same set of laptop gestures: horizontal and vertical tilt, and physical hits to two different locations on the laptop. The composition uses the Wekinator to detect these gestures using three support vector machine classifiers. The first classifier classified performers' hits or taps on the laptop into two discrete hit locations at the side and top of the machine, as well as into "no hit" and "tilting" states. All performers used the same set of hit classifiers. The second and third classifiers quantized horizontal and vertical tilt, respectively, into

Table 7.5: The four laptop instruments played in *G*. Columns indicate the instrument name, the ChucK classes used for synthesis, and the mapping from vertical tilt, horizontal tilt, and hits at two locations on the laptop to the parameters of the sound.

Name	**Synthesis Algorithms**	**Mapping**
Wobble Bass	PulseOsc	Vert. tilt: "Wobble frequency" Horiz. tilt: Pulse width Hit 1: Sidechain compressor Hit 2: "Frequency jump effect"
Lead Synth	PulseOsc TriOsc Saxofony	Vert. tilt: High-pass cutoff frequency Horiz. tilt: Pulse width Hit 1: Start "howl effect" Hit 2: Stop "howl effect"
Bells	BandedWG	Vert. tilt: Chorus effect mix level Horiz. tilt: Rhythm patterns of "Fast forward" effect Hit 1: "Fast forward" effect Hit 2: Variation of "Fast forward" effect
Drums	SinOsc ResonZ Shakers Noise ModalBar	Vert. tilt: Select drum sounds used Horiz. tilt: "Bitcrush effect" Hit 1: Play drum 1 of selected set Hit 2: Play drum 2 of selected set

3 or 5 discrete classes, depending on the instrument. The three classifiers' outputs were used to control different ChucK synthesis classes depending on the performer's instrument, as described in Table 7.5.

To perform classification, the Wekinator used the built-in motion feature extractor alongside a custom ChucK feature extractor written by Weitekamp. This custom extractor computed additional features from the laptop's accelerometers, including multiple time-averaged derivatives and maxima. The tilt classifiers used only the built-in motion feature extractor, and the hit classifier used only the custom features.

The following discussion of *G* draws on an in-person interview conducted with Weitekamp on 23 April 2010 and e-mail correspondence in October 2010.

7.4.1 Composing the Piece with the Wekinator

Background

Prior to creating *G*, Weitekamp had considerable expertise developing music performance software in ChucK and Processing. He had developed software for use by both himself and other musicians; notably, Weitekamp had previously written and publicly distributed "SmackTop," a software package for translating a performers' laptop

"smacks" into MIDI messages (Weitekamp 2009). The SmackTop package was implemented without using machine learning or the Wekinator, and prior to beginning work on *G*, Weitekamp characterized his familiarity with the Wekinator as "1" and his familiarity with machine learning as "2" on a scale from 1 (Not at all familiar) to 5 (Extremely familiar). Weitekamp did not take part in the participatory design seminar discussed in Chapter 4, and did not receive instruction on the Wekinator in the PLOrk class discussed in Chapter 5 (he was enrolled in a different section of the course, COS/MUS 316).

The motivation for using the Wekinator in *G* stemmed from Weitekamp's desire for a larger gestural vocabulary than that offered by SmackTop, and a potential to customize the vocabulary in a way that was appropriate for the performers: "I was interested in the challenge of classifying different types of 'hits' using Wekinator. For example, using the algorithm to classify between hitting the right and left sides of the laptop, or soft vs. hard hits. I really wanted the performers to be able to choose their own gestures to perform a fixed musical composition." As we discuss below, he did not ultimately find it feasible to allow performers to design customized gestures, but he writes, "Though this wasn't realized, I feel that with more preparation time it would have been possible to do play along learning as the prelude to the piece. Some of the motivation for allowing the performers gestural freedom comes from my experience performing with PLOrk. Many of the hard-coded, linear mappings between controller and instrument lead to awkward physical gestures that were either uncomfortable or unrealizable for some of the performers. As well, there were certain pieces where I felt my gestures were more of a gimmick than an expression. I wanted to develop a piece which would remain fixed musically, but allow each performer to create an individual gestural mapping that was both comfortable and expressive."

Using the Wekinator

Weitekamp's work with the Wekinator entailed some research to determine which custom features he might compute from the raw built-in accelerometer outputs in order to best discriminate among different hit locations. Over the semester, he developed his own set of Wekinator-trained models for hit classification in conjunction with refinements to his custom feature extractor.

As mentioned above, Weitekamp's original plan for the piece was to have performers define their own gestures to use in performance. Though Weitekamp himself was able to use his final set of features to train a hit classifier capable of reliably discriminating among four different hit locations, he was disappointed to find that, after several weeks of rehearsal in which performers used playalong learning to train their own models, not all of his performers were consistently successful in creating classifiers that were accurate enough for use in performance. Therefore, for the final performance, he chose to remove the playalong component of the piece. Instead, he distributed pre-trained models to the performers a few weeks before the performance. These models were trained by him to recognize just two hit locations, and performers could practice with them outside of the performance to ensure they could consistently perform hits that were distinguishable by the classifiers.

Early in the process of developing the piece, Weitekamp contacted us and expressed that he was having difficulty creating training data for the different hits. In particular, it was hard to synchronize his hitting of the laptop precisely with a playalong score or with manually clicking on the GUI button to start and stop recording, so that training features extracted just before, during, and just after a laptop hit were labeled cleanly. To help solve this problem, we implemented the OSC control mechanism for starting and stopping recording, described in Section 3.5.2. Weitekamp then wrote OSC "gating" code in ChucK, which he used during training of the different hit locations. The gating mechanism enabled recording of new training examples only when the internal accelerometer values exceeded a hard-coded threshold (i.e., immediately after a laptop was hit). With this mechanism in place, Weitekamp was able to successfully train accurate classifiers.

Interactions During Model Creation

When creating the training data for the models he built himself, Weitekamp used playalong recording as well as the standard training data recording mechanism. He focused his effort on building classifiers that "classified the different hits correctly and accurately, and [were] not sensitive to laptop orientation." To improve models whose performance Weitekamp characterized as "loose" (meaning that hits that seemed similar to the training examples did not result in accurate classifications), Weitekamp added more training examples. Before getting the gating mechanism to work well, he often used manual editing of the training data in the spreadsheet editor, in order to delete all but the training data produced immediately after a hit was performed. He very frequently deleted the entire training set and started building a model from scratch, "Especially when I had to update the models or change input features, or if classification was wrong." As mentioned above, Weitekamp experimented extensively with using feature selection to discover which of his custom features were most useful for classifying different types of hits.

Weitekamp states that he never changed the classifier algorithm or its parameters in an attempt to improve the models. He started by using a support vector machine algorithm for classification, and it just "seemed to work... and I had too many other things to optimize."

When asked if he became better at using the Wekinator at he composed, Weitekamp replied, "Certainly. Primarily, I became better at providing consistent examples to the models. This was a hard skill to try to teach my performers... We spent a lot of time learning how to 'teach' the computer. I also feel that I became better at thinking 'like a computer'; once I was able to understand which input features were important, I felt I could better prepare the input data to optimize classification." Weitekamp describes his approach to training the models as conscientious: "In order to train my model successfully, I needed to sit very still and give the machine consistent inputs during the play along learning." He seemed confident that, given more rehearsal time and experience with the Wekinator, his performers could have also learned to use the Wekinator more effectively to create classifiers for their own gestures. In practice, the need to teach performers to use the Wekinator effectively

ended up conflicting with the need to simultaneously teach performers how to play the piece; during the rehearsal process, performers communicated to Weitekamp that it was important to them to have access to a set of classifiers that they could use to practice their part of the piece and play effectively in rehearsal, even before they possessed the expertise to successfully create customized models for themselves.

7.4.2 The Wekinator's Influence on the Composition

As illustrated in Table 7.6, although Weitekamp began the composition of *G* with specific ideas about the gestures and sounds he wanted to use in the piece, he changed his mind about these aspects of the composition during his work with the Wekinator. The gestural vocabulary he ultimately used in the piece was highly informed by the gestures that he found to be easy to classify, and that rehearsals revealed to be robust across different performers and laptops. He wrote, "I quickly found that certain types of hits were not easy to classify. For example, classifying a hit on the right vs. left side of the laptop is not trivial—most likely because it only involves one accelerometer axis. For the final performance, I used one side hit and one top hit as the training examples."

As discussed above, the failure of his original plans to teach performers to successfully train their own models led to a significant change in the character and performance technique involved in the piece, when Weitekamp decided to have the performers use only classifiers pre-trained by him.

Additionally, Weitekamp believes that the sonic character of the piece was influenced by his experiences developing the gesture classifiers, and by testing out many different mappings to find one that was suitable: "I developed the Wekinator code well before designing the instruments or composing the final score. As a result, most of the ideas for percussive events or tilt-controlled FX were either directly or subconsciously influenced by Wekinator. Because the gesture-sound mappings had to be both robust and interesting, it took a number of iterations to develop a good combination."

Finally, Weitekamp describes the Wekinator as an essential force motivating the constraints he chose to shape the composition. "The Wekinator provided a new tool for employing the laptop as an expressive controller. For this composition, I chose to use no additional hardware because I wanted to explore the limits of this idea."

7.4.3 Evaluation of the Wekinator

As illustrated in Table 7.7, Weitekamp evaluated the Wekinator as being highly valuable in creating mappings more easily, creating more expressive mappings, and creating a new type of composition using a new type of compositional process. Asked to specifically describe how the Wekinator enabled a new approach to composition, Weitekamp responded, "Wekinator allowed me to turn the laptop into both a percussive and graduated controller (simultaneously). Because the gestural mapping between the inputs and outputs need not be linear, the process of sonifying a gesture was easier and ultimately lead to a more natural mapping."

Table 7.6: Rated level of agreement with statements about compositional goals before and during work with the Wekinator in *G*, rated on a 5-point Likert scale from 1 = "Strongly disagree" to 5 = "Strongly agree."

Statement	**Agreement**
Before I started working with the Wekinator...	
I had a specific gestural interface in mind that I wanted to use or build for the piece	5
I had specific physical gestures I wanted to use in the piece	3
I had a specific palette of sounds I wanted to use in the piece	4
I had specific ideas about how I wanted gestures to control sounds in the piece	5
While working with the Wekinator...	
I changed my mind or had new ideas about the gestural interface(s) that I wanted to use, or that I wanted to build	5
I changed my mind or had new ideas about the physical gestures I wanted to use in the piece	5
I changed my mind or had new ideas about the palette of sounds I wanted to use in the piece	4
I changed my mind or had new ideas about how I wanted gestures to control sounds in the piece	4

When asked what aspects of the Wekinator were most helpful in composing and performing the piece, his first response was "Every aspect, because it was integral to the concept of the piece." He then added that the OSC gating mechanism was particularly useful for recording clean training data for different laptop hits.

Even though Weitekamp was a proficient software developer, he indicates that he would not have been able to create this piece by hard-coding the analysis of the features and their mapping to sound. In his opinion, the accelerometer analysis and classification would have been exceedingly difficult, awkward, and time consuming, if not impossible, to perform by writing code.

Weitekamp continues to be excited by the Wekinator's potential to allow computer music performers to customize their performance interfaces. Having participated in a performance of *CMMV* (Section 7.2), he appreciated that supervised learning could be used to create new instruments with greater expressive possibilities than simple, linear, and one-to-one mappings. At the same time, he desires to create expressive instruments that do not lose all of the transparency that such simpler interfaces offer to a performer learning how to play them, and he is interested in how the Wekinator could be used by performers to create their own mappings or to customize mappings created by a composer (e.g., by adding new data or changing the model in minor ways). One potential application of the Wekinator that he proposed involves

Table 7.7: Rated level of agreement with statements about the Wekinator's usefulness, in G; rated on a 5-point Likert scale from 1 = "Strongly disagree" to 5 = "Strongly agree."

Statement	**Agreement**
The Wekinator allowed me to create mappings between gesture and sound more easily than other techniques.	5
The Wekinator allowed me to create mappings between gesture and sound that were more musically expressive than other techniques	5
The Wekinator allowed me to create a kind of music that isn't possible or that is hard to create using other techniques	5
Using the Wekinator allowed me to approach the process of composition in a new way	5

having performers use playalong recording to construct their own training sets, while watching a video of the composer playing along to the same sounds: this would allow performers insight into the composer's training data creation process, and it would allow them the freedom to deviate from the composer's training process for aesthetic or practical reason (e.g., having shorter limbs).

Weitekamp composed G using a version of the Wekinator that did not yet have the graphical editor, and when he was shown the graphical editor he saw it as potentially greatly useful to informing the user about the nature of the training data and speeding up the model-building process. He is interested in having additional capabilities to visualize or otherwise understand how the models were working and how a user might most meaningfully change the learning algorithms or parameters to achieve different results. Weitekamp has also expressed interest in making the Wekinator more available to a wider variety of electronic musicians, through integrating it with a future version of the SmackTop software, and by enabling it to output and receive MIDI for easier use by musicians who do not use OSC.

7.5 Discussion

7.5.1 How the Wekinator was Used

The composers discussed in this chapter applied the Wekinator to the creation of diverse and distinctive new musical instruments and compositions. In the works by Trueman, Nagai, and Weitekamp, the Wekinator was used to create instruments from commodity computer input devices, from a piece of tree bark containing custom light-sensor circuitry, and from the laptop's built-in sudden motion sensors. Nagai and Trueman created instruments with continuous gesture-to-sound mappings, and Weitekamp created instruments incorporating discrete gesture classifiers. Nagai's instrument was performed solo, and Trueman's were performed within a heterogenous ensemble of acoustic and laptop performers; Weitekamp used the same gesture clas-

sifiers within all four different laptop instruments used in his composition for laptop ensemble. The compositional styles of these pieces also spanned a wide range, from avant-garde to electronica-inspired and rhythmic.

Each composer developed distinct strategies for interacting with the Wekinator and for integrating it into his or her larger compositional process. Nagai experimented with many different neural networks for creating mappings for her instrument, and she developed the sound synthesis component of her composition in conjunction with the development of the mappings. She only finalized her instrument's mapping a few days before the performance of her piece. Trueman, on the other hand, gradually refined his mappings over time, and he used the Wekinator to create mappings for hardware controllers and synthesis algorithms that were relatively fixed. Trueman completed his mappings before rehearsals of his piece so that performers could learn how to play them, and through three performances of his composition, he has kept the mapping of one instrument fixed while continuing to change both the mapping and the hardware controller for the other. Weitekamp used the Wekinator to experiment with different gesture classifiers while at the same time refining and improving his feature extractor, and throughout the rehearsal period for his piece, he experimented with both allowing performers to train their own classifiers and with training classifiers himself for them to use. Both Trueman and Nagai developed strategies for creating training data that would produce mappings that balanced predictability and control with surprise and complexity, while Weitekamp developed a strategy for creating training data that resulted in classifiers whose behavior was as predictable and robust as possible.

Despite these differences, each of the instruments created by these three composers shares a common focus on the expressive use of the human body, and each composer saw the Wekinator as a valuable tool in enabling him or her to prioritize expression and physicality throughout the activities of composition, instrument-building, and performance. Nagai and Trueman both expressed that, in the process of composing and instrument building, it was important to establish a mode of performance in which the relationship between performer gesture and sound was tightly coupled, not simple or linear, and offered an range of expressive possibilities that a performer could learn—with practice—to control in a musically sensitive manner. When the instruments they created were successful in meeting these criteria, the composers likened them to acoustic musical instruments. In other words, they used the Wekinator to create new performance interfaces that possessed certain musically essential qualities of conventional, existing instruments, but which—through their use of new controller interfaces and synthesis algorithms, and the ability to tailor their design to a particular performer or composition—transcended certain physical or compositional limitations of such instruments. For Weitekamp, the Wekinator was the inspiring force behind his attempt to create a composition wherein each performer had the freedom to design his own expressive gestural vocabulary. While this vision was not realized in the performance, his ultimate use of the software still enabled him to efficiently build a set of laptop instruments whose performance technique demanded performer-laptop interactions that were physically engaging to the performers and visually engaging to the audience.

We further discuss how the Wekinator supported this type of work, and the consequences for future use of machine learning in musical and non-musical applications, in Chapters 8 and 9.

7.5.2 Useful Features of the Wekinator

The working process of all three composers took advantage of the Wekinator's functionality for iteratively creating and editing training data, building models, experimentally running the models in real-time, and modifying the supervised learning problem, data, and/or other aspects of their instruments or compositions in order to improve the models. Some of the aspects of the Wekinator that allowed them to perform this work successfully included its capabilities for users to switch between these different tasks with a low amount of overhead, to efficiently provide training data, to frequently evaluate models by interacting with them in a real-time manner similar to the way in which they would be used in performance, and to perform all these tasks without the need to write code or become machine learning experts.

The Wekinator's ability to use OSC to receive features and control messages and to send synthesis parameters was also particularly useful to all composers, and this ability enabled composers to apply the Wekinator alongside the compositional software tools they already knew how to use and preferred.

The particular learning algorithms built into the Wekinator were also useful to composers. In Chapter 9, we discuss how neural networks functioned as a useful compositional tool for composers discussed here and in Chapter 4. Support vector machines were useful to Weitekamp because they were capable of accurately classifying the input gestures of interest to him.

7.5.3 Influence of the Wekinator on Composition

Each of the composers agreed or strongly agreed that, while they were working with the Wekinator, they changed their minds about the gestures, sounds, and gesture-sound mappings that they planned to use in their instrument or composition. Occasionally, these changes of course were a result of working around barriers that the software presented to their achieving their initial goals: for example, Nagai took many measures to reduce the training time of her neural networks, including limiting the number of light sensors that influenced each sonic parameter, and Trueman ended up crafting his Tethered-uBlotar playalong score to reside within a narrow sonic range to avoid the "pointy" mappings that were most easily produced from playalong recording. Sometimes, composers discovered through their use of the Wekinator that their initial goals were simply infeasible: Weitekamp discovered, for instance, that it was too hard for him to train all of his performers to produce consistently good classifiers given the practical and compositional constraints of his piece.

At the same time, composers often relied on their experimentation with the Wekinator to discover new sonic and gestural possibilities that they had not previously imagined, and they adapted aspects of their instruments or compositions to take

advantage of these possibilities. The Wekinator also inspired composers to consider new composition and performance paradigms, such as the composition of complex mappings or the engagement of performers in the instrument building process, which would not have been possible—or which just seemed too impractical to be considered—using other tools.

7.6 Conclusions

In this chapter, we have described three new musical compositions that have been completed by Wekinator users. The success with which these users have been able to apply the Wekinator to their work demonstrates that, while various improvements could still be made to the software, the Wekinator in its current form is a useful tool for real-world applications. Furthermore, users having low low familiarity with machine learning were able to apply it successfully to their work.

The diversity of ways that these users applied the Wekinator in their work also attests to its flexibility as a compositional tool. The Wekinator's OSC capabilities, which allow it to be used in conjunction with arbitrary feature extractors, synthesis algorithms, and visualizations, were crucial to enabling these users to employ the software in conjunction with other software and hardware systems of their choosing. Different users also took advantage of different mechanisms for interactively applying supervised learning to their work, including different mechanisms for recording and editing the training data (e.g., playalong recording, OSC-gated recording, and editing using the spreadsheet editor) and for modifying the supervised learning problem (e.g., changing the classifiers, changing feature selection, and adding meta-features).

Furthermore, the Wekinator invited users to imagine vastly different applications, from performer-customizable mappings to turning a piece of tree bark into an expressive instrument, each of which incorporated a unique musical aesthetic and presented distinct technical challenges. While each of these applications was different from those we had imagined when designing the software, the Wekinator was able to inspire these projects and function as a useful tool for tasks that would have been impossible or difficult for users to accomplish otherwise. Additionally, the Wekinator supported each of the composers in creating music that privileged the expressive use of the human body during both composition and performance.

The composers discussed in this chapter have been extremely helpful in informing us about the features of the Wekinator that are important to users, the features that they would like to see added to the software, and the potential future applications they envision may be possible with the Wekinator. We intend to continue to collaborate closely with them and other Wekinator users to drive improvements to the software and cultivate ideas for future research directions.

Chapter 8

Discussion: Interacting With Supervised Learning

8.1 Introduction

In each of the last four chapters, we presented a study of a user or set of users applying interactive machine learning to their work in computer music. Users' projects included the creation of continuous gesture-to-sound mappings in the design of new, musically expressive digital instruments; the recognition of customized, discrete gestural vocabularies for controlling sound; and the classification of standard cello bowing gestures. In these projects, the goal of the supervised learning system ranged from being constrained by musical conventions to being completely open to creative design by the user.

Across this range of musical applications, all users took advantage of the Wekinator's capabilities to interact with the supervised learning process. Users rarely employed a workflow typical of conventional machine learning applications, involving the application of standard metrics of generalization accuracy to choose among candidate models trained on the same dataset. Instead, users interactively created and edited the training data, they interactively evaluated trained models by running them in real-time, and they iteratively refined their models by modifying the data or algorithm, evaluating the trained model, and repeating until they were satisfied with the outcome.

In this chapter, we will examine findings across all four user studies to discuss the important roles that interaction—encompassing both human-computer control and computer-human feedback—played in the development of supervised learning systems, and to discuss our findings regarding the differences between interactive and conventional machine learning contexts. We will also discuss the implications of our findings regarding the requirements and challenges in the analysis and design of algorithms and interfaces for interactive supervised learning, with attention to how the characteristics of the computer music problem domain shaped the types of interactions that were possible for the users we observed.

We begin this chapter with a review of how users in the four studies employed interaction in modifying the training data and feature selection, changing algorithms and parameters, and evaluating the supervised learning systems. We discuss the roles that evaluation played in enabling users to assess models against objective and subjective criteria, in providing feedback about the efficacy of users' actions, and in shaping users' goals. We then discuss the role that generalization accuracy plays in interactive supervised learning, which we propose may be different from its role in conventional machine learning. We also discuss the role that training set size plays in interactive machine learning and raise the possibility that interaction might serve to decrease sample complexity.

We then present our observations of the characteristics of algorithms and interfaces that impacted users' experiences with the Wekinator, and we outline a framework for understanding algorithm and interface analysis and design in terms of affordances, usefulness, and usability. Last, we enumerate the characteristics of the computer music domain that were key to enabling the types of interactions that we observed, and we discuss how the findings of our work might be generalized to other domains.

8.2 Interactively Modifying the Learning Problem

In all four studies, users employed interactions with the training set—including creating data, adding new data, and deleting and editing data—more frequently than interactions that modified other aspects of the learning problem, such as editing the learning algorithm or its parameters, or changing the features selected for a model. The training dataset itself can be interpreted as an important *interface* through which users exercised control over the supervised learning problem. Significantly, interactions with the training set often enabled users a direct and effective means of exercising control over aspects of the learning problem that were important to them, but which could not always be directly controlled solely through the conventional machine learning interactions of changing the learning algorithms or their parameters. In this section, we review how users employed interactions with the training set to accomplish their goals. We also review how users employed feature selection to modify the learning problem.

8.2.1 Creating the Training Dataset

In all applications studied in this thesis, users created the entire training datasets interactively. The choice of what training data to create reflected fundamental characteristics of the learning problem, and we observed that users developed conscious strategies for creating training datasets that best helped them accomplish their goals. In particular, the training set was used to define the scope of the learning problem, to communicate the essential characteristics of each class, to preemptively minimize error and cost of the trained model, and to sketch out areas of interest and denote boundaries of the input and output spaces to be used.

First, a user's choice of training examples communicated the desired scope of the learning problem. One component of scope is the number of classes assigned to instances in the training set. The number of gesture classes was varied by PLOrk students and the K-Bow cellist to adjust the computational difficulty of producing a working classifier and to adjust the musical usefulness of the trained model (e.g., the number of sounds it could be used to control). Another component of scope concerns the dimensions of input variability to which a model should be robust. For example, the K-Bow cellist created training datasets for most gesture problems by demonstrating each gesture across all four instrument strings, and the training set typically included gestures performed within a limited range of bowing speeds that excluded extremely fast and slow bows. By exhaustively varying the set of strings but not the range of possible tempos present in the data, she created classifiers that could perform well for gestures on different strings, but that might fail at extreme tempos. The choice of scope may reflect a choice about tradeoffs between creating models with limited power that are easy to create and creating more expressive models that may require more training examples and more fine-tuning of other aspects of the learning problem. We observed users dynamically changing the scope represented by their datasets in order to build models that were both feasible to construct and that effectively met their musical goals.

Second, several users consciously employed training data for classification problems to communicate what they felt to be essential characteristics of each class, or "prototypical" gestures of each class. Commonly, this was done by providing input gestures that were as free as possible from non-essential variations in the input features. Many users reported learning through their use of the Wekinator that minimizing noise and unnecessary variation in the training examples led to better classifiers.

Third, some users' strategy for creating classification training examples involved explicit attention to designing classifiers with low misclassification error or cost. For example, several PLOrk students (who had the ability to design their own gestures for classification) defined their set of gestures according to what they thought the Wekinator would be able to easily classify. Often, this amounted to maximizing the inter-class variance of the gestural features. Another PLOrk student consciously defined her gesture set so that the feature values that were easiest for her to produce corresponded to the classes she expected to use most often during performance. These actions reflect the fact that users created training data not just to represent the abstract concept they intended the model to learn, but according to assumptions and intuition about which characteristics of the dataset would affect the trained classifier.

Fourth, in continuous regression problems, many users discussed in Chapters 4, 5, and 7 used the training dataset—especially the first training dataset created—to provide a "sketch" of the learning problem intended to produce a simpler model than the one they ultimately planned to build. For some users, this sketch was constructed to denote both areas of interest and limiting boundaries in the spaces of input gestures and output sounds that they intended the model to use.

Importantly, the ability to delete the entire training set and create a new training set from scratch enabled users to employ the training set as an interface to express

changes in the learning problem scope, class definitions, cost, or other priorities, when these changed throughout the use of the system. Furthermore, even when these properties of the intended learning problem did not change, many users found that their strategies for employing the training set to accomplish their goals evolved as they gained more experience with the system. Deleting and re-creating the training data enabled them to employ their new strategies.

8.2.2 Creating Training Data Through Playalong

Some composers discussed in Chapters 4 and 7 frequently used the playalong interface for creating training data for their neural network mappings. Like the training examples created without the playalong interface, the examples created using playalong also reflected users' choices of scope and the intended sonic and gestural boundaries of a mapping. Furthermore, users employed the playalong interface to create data that encoded more fine-grained information about the desired model function. Whereas creating training data without playalong involved decisions about which static gestures (e.g., joystick or tether positions) would correspond to which static sounds, composers used the playalong interface to create data that captured more fine-grained aspects of their ideas for how sound should vary with performer motion.

Additionally, composers used playalong to engage their own musical and physical expertise more deeply in the model-building process. For one composer in Chapter 7 (Trueman), the playalong interface was important because it allowed him to practice the gestures along with the sound before creating training examples. The training examples he ultimately used therefore represented a gesture-to-sound mapping that he *felt* was appropriate, and that he felt comfortable using. In this way, playalong enabled composers to create mappings that had already been roughly evaluated for how they might be used in expressive, real-time performance, and that had been judged to be suitable. In contrast, users who did not employ playalong had to train the models and run them in order to make such judgments.

Not all composers used playalong frequently. It could be that composers who did not use playalong were not as concerned with managing the ways in which sounds and gestures co-varied over time (at least within the projects they were working on). Or, it could be that they were content with how the Wekinator managed these characteristics for them when they didn't use the playalong interface. This seemed to be the case for one composer discussed in Chapter 7 (Nagai), who indicated that she developed a slow and subtle gestural vocabulary for playing her Wekinator-based instrument because she found such gestures to be most expressively effective within the mappings the Wekinator generated.

8.2.3 Incrementally Modifying the Training Set

Adding More Training Data

When composers and students in chapters Chapters 4, 5, and 7 wanted to improve or change a trained model, they often found it most convenient or useful to delete the

entire dataset and begin creating new examples from scratch. However, users in each of those studies, as well as the K-Bow cellist, found adding data to an existing training set to be useful under certain circumstances: to correct errors, to take advantage of desirable output values, and to make models more complex.

First, data was sometimes added to correct model errors. For example, when the K-Bow cellist discovered that a trained classifier made a mistake on a gesture, she often added more training examples created by demonstrating that gesture and supplying the correct label.

Second, users employing neural networks for gesture mappings sometimes added more training examples to take advantage of new sounds that they discovered; in this case, the construction of the new examples involved deciding which input gestures should produce the desired synthesis parameter set. Sometimes, composers discovered new sounds they liked by typing new parameter values into the Wekinator interface or by manipulating an external synthesis environment GUI (e.g., a Max/MSP patch being controlled by the Wekinator). By adding these parameter sets to the training set, along with new gestural feature vectors, composers aimed to make these sounds available in the instrument mapping. Additionally, sometimes composers discovered new sounds that they liked while playing with a trained mapping in real-time, and they desired to make these sounds more prominent or more easily playable in the instrument. In this case, they accomplished this by adding more training examples associating that sound with additional gestural inputs.

Third, users employing neural networks for gesture mappings sometimes added more training examples in order to produce more complex mappings. For example, sometimes composers desired to broaden the range of sounds capable of being produced by the mapping. They accomplished this by adding examples in which the synthesis parameter values deviated from the types of values output by the current trained models. As another example, sometimes composers desired to create mapping functions in which the sound changed in more complex ways with changes gesture, and they accomplished this by creating new examples at locations in the gestural input space where they wanted to insert complexity into the mapping.

Deleting Training Data

While it was more common for users in all studies to delete the entire training dataset and re-create it than to delete only a portion of the training set, sometimes a subset of training examples were deleted to undo changes to a model, to remove errors from a training set, to reduce training time, or to attempt to incrementally change a model's behavior in other subtle ways.

Using the spreadsheet editor, users could delete all training examples recorded in a single training round. Commonly, this was done to effectively "undo" the changes recently made to a model and restore it to an earlier state.

Users also sometimes deleted examples that were suspected of containing errors or likely to be noisy. Several PLOrk students reported using the spreadsheet editor to visually identify examples that appeared to be outliers, then delete them. In building the K-Bow gesture classifiers, the graphical editor was occasionally used to manually

remove class labels from certain training examples, for example when it was clear from the visualization that a feature vector represented a bow movement that was in between up- and down-bow gestures.

Occasionally, PLOrk students and composers used selective deletion to simplify the learning problem or to reduce the training time of the learning algorithms. For example, one composer in Chapter 7 (Nagai) reduced her training set sizes to reduce the training time of her neural networks, which she found to be frustratingly long when she had many parameters, many features, and many data examples. Another composer in Chapter 4 wrote that "having too much data seemed to cause the model to even everything out to a gray middle ground, so nothing would change." By deleting a portion of his examples, he obtained a model with more varied outputs.

Editing Training Data

Users very rarely employed the spreadsheet editor or graphical editor to change the feature or parameter value of existing training examples. Our observations seem to indicate that in most applications, given the Wekinator's available interfaces for creating and editing data, re-creating the training examples was a more efficient and accurate means of modifying the training set compared to making manual edits. Manual editing was only commonly done in the K-Bow project, in order to enable the cellist to first create the training examples by using natural performative gestures, in which she switched fluidly between gesture classes (e.g., up-bows and down-bows) without pausing to interact with the GUI. The graphical editor was then used to manually apply labels to the recorded gesture features. This enabled greater labeling precision than using a playalong score, because gesture beginnings and endings could be labeled precisely without requiring the cellist to synchronize with a score. However, this type of interaction was only feasible for gestures classes such as Bow Direction where the gesture class was clearly distinguishable within the graphical editor's visualization of the input features.

8.2.4 Editing Feature Selection

Feature selection—the choice of which of the available input features would affect which of the trained models—was employed by some composers and students and by the K-Bow cellist as another means of modifying the learning problem. Like modifications to the training dataset, different feature selections were applied iteratively over the course of working with the system: it was common for users to try one feature selection setting, train and evaluate a model using that setting, then change the feature selection later and evaluate the outcome. Feature selection was employed by users to improve classification accuracy, reduce training time, and enforce independencies.

In conventional (non-interactive) machine learning, it is common to experiment with selecting different subsets of the available features to use in creating a model. Certain learning algorithms are particularly sensitive to features that are redundant or that are irrelevant to the learning problem, so removing these features can allow for the construction of more accurate models (Guyon and Elisseeff 2003; Fiebrink

2006). Feature selection was used for this same reason in creating the K-Bow gesture classifiers. Some features were clearly irrelevant to certain classification problems (e.g., the "grip" sensor was not relevant to the Bow Direction problem) and they were unselected. For other problems, several available features presented somewhat redundant information; for example, the minimum, maximum, and mean hair features were all relevant to the "On/Off String" classification problem, but only the mean was used to train the classifier.

Second, in her composition discussed in Chapter 7, one composer (Nagai) used feature selection to shorten the training times of her models. By using fewer features, as well as by limiting her number of training examples, her set of neural networks became faster to train and easier, overall, to use.

Finally, composers and students in Chapters 4, 5, and 7 sometimes used feature selection to enforce independencies between certain gestural dimensions and certain behaviors of the trained model. For example, a few PLOrk students experimented with building mappings where the x-axis of the built-in laptop accelerometer controlled one aspect of the sound only, and the y-axis controlled another.

Several composers and students—though not all—seemed to naturally imagine mappings in which certain gestural dimensions were constrained to affect certain compositional dimensions, and they found it important to be able to construct these types of mappings in the Wekinator. Composers in the participatory design process were frustrated by the Wekinator's inability to enforce these independencies before feature selection was added to the software. Also, PLOrk students who did not understand what the feature selection mechanism of the Wekinator did, yet who desired to create these types of mappings, went to great lengths to construct the training dataset in a way that would allow them to approximate these independencies without explicit feature selection.

8.3 Editing Algorithms and Algorithm Parameters

Users in all four studies changed the learning algorithms or their parameters much less frequently than they modified the training data or features. Composers and students creating continuous gesture-to-sound mappings only used neural networks (the only algorithm available to them), and they almost never changed the network architecture or learning parameters, though these actions were possible. One composer in Chapter 4 experimented casually with changing the network learning parameters, but he had difficulty setting the parameters to values that meaningfully and predictably affected the outcome of training (e.g., by affecting training time or the quality of the trained models).

Editing the algorithms and their parameters was sometimes useful, though, for PLOrk students, composers, and the K-Bow cellist in building discrete gesture classifiers. Most frequently, users experimented with changing the classifier algorithm when they observed that the current classifier produced a poor model. This was especially

apparent in the PLOrk study, where the default algorithm (AdaBoost boosting on decision stumps) performed poorly for many multi-class problems of interest to the students. Different classifiers were also used in the K-Bow study to obtain more accurate models.

The parameters of classifier algorithms were almost never changed. In the PLOrk study, only two students ever tried changing parameters of an algorithm. In the K-Bow study, the k parameter of a k-nearest neighbor (kNN) classifier was increased to produce a smoother decision boundary.

8.4 Interactively Evaluating Trained Models

The Wekinator offered two mechanisms for evaluating trained models: computing objective metrics (training accuracy and cross-validation accuracy), and directly evaluating the models by running them on real-time inputs and visually or auditorily observing their real-time outputs. Users employed interactive, direct evaluation more frequently than computing cross-validation or training accuracy, but some users still found the objective metrics useful. In this section, we will discuss how users employed both types of accuracy to judge trained models against a variety of objective and subjective criteria, and how they used the outcomes of evaluation to inform their subsequent actions with the system.

8.4.1 The Use of Direct, Interactive Evaluation

Direct evaluation was the most frequent method of evaluating trained models used by the PLOrk students and the K-Bow cellist, and it was the only evaluation method that composers used in the work discussed in Chapters 4 and 7. The composers and PLOrk students in Chapters 4, 5, and 7 all directly evaluated trained models by listening to how the models' outputs drove synthesis parameters as they demonstrated real-time gestures, and the K-Bow cellist evaluated models by visually examining their output values and posterior distributions in the Wekinator or in a Max/MSP visualization. When users experimented with models in real-time, they evaluated the models against a variety of criteria, including correctness, cost, decision boundary shape, confidence, complexity, and unexpectedness. Users employed direct evaluation both to evaluate whether or not a model was good enough for use in performance (i.e., good enough to stop the interactive machine learning process) and, in the case that a model was not yet good enough, to help them decide how to take action to improve the model.

Next, we summarize common evaluation criteria employed by users and how users took action to improve the model against these criteria.

Correctness

Predictably, direct evaluation was employed to systematically check a model's correctness over a range of the input space of gestures. In all studies—including the design of new, expressive musical instruments, where there was no objectively right

or wrong model behavior—users identified a model's behavior as incorrect when it produced an output contrary to what they believed was appropriate and expected. As discussed earlier, users often addressed incorrect model behaviors by adding new, correctly-labeled examples to the training set, or by deleting the whole training set and re-creating it.

Cost

Our observations suggest that users held an implicit error cost function that variably penalized model mistakes based on both the type of misclassifications (i.e., the user's label and the model's label) and their locations in the gesture space. For example, the K-Bow cellist verbally indicated that classification mistakes a human cellist might easily make were less problematic. Another concern for PLOrk students and composers was whether a model could produce desirable or correct outputs for the types of gestures that would be used in a performance; the model's behavior on gestures not used in performance was inconsequential. Users expended more effort fine-tuning model behavior on gestures likely to be used in performance by adding to and re-creating the training dataset.

Shape Decision Boundary and Posterior Distribution

As discussed in Section 6.5.3, the shape of a classifier's decision boundary was sometimes important to the K-Bow cellist. It was somewhat more important to her that the switch from one gesture label to another happen smoothly as she varied her gesture, than that this label switching happen at a precise location in the gesture space. Actions taken to smooth jagged decision boundaries included changing algorithm parameters (e.g., increasing k in kNN) and adding smoothly-labeled training data along the boundary area.

As discussed in the same section, the K-Bow cellist also took model confidence into account when evaluating its quality. She reacted unfavorably when a gesture model classified her gesture correctly but also assigned relatively high posterior probabilities to several incorrect labels, and she sometimes attempted to improve the model's confidence by adding more training data. She judged more "certain" classifiers as being of a better quality than less certain ones, and she also valued posterior distribution shapes that had predictable behaviors that she could use in post-processing (e.g., to look for "signature" distribution shapes that signaled likely misclassifications).

Complexity and Unexpectedness

Finally, composers and students discussed in Chapters 4, 5, and 7 evaluated models to assess whether they produced sonically interesting parameterizations of the synthesis algorithm, which a human could manipulate over time in a musically sensitive way. Most composers and students using the Wekinator to produce continuous mappings typically did not construct the models with a full set of physical and musical gestures already in mind; rather, a common strategy for building the models was to choose a

few different and interesting synthesis parameter values that they wanted the instrument to be capable of playing, match each of these with a different gestural input in the training set, then directly evaluate the trained model to discover and explore the sounds that arose as they moved between and around the gestures present in the training set.

When evaluating the model using gestures outside the training set, two of the characteristics that were most important to users were its complexity and unexpectedness. Unlike in a conventional machine learning application, complexity and nonlinearity were desirable, and composers sometimes added training data with the explicit intention of making the model more complex. Some composers remarked that the complexity of functions generated by the Wekinator's neural networks made their new instruments feel more like traditional, acoustic instruments, which by nature involve very complicated relationships between the physical gestures of a performer and the sound produced. Additionally, users valued the fact that they could be surprised by the sounds generated by a gesture not in the training set, allowing them to find unimagined and compositionally useful sounds in the synthesis space. As discussed above, users frequently modified the training dataset to make models more complex, take advantage of new sounds, and reflect changes in their choice of the range of gestures and sounds they wanted to use.

8.4.2 The Use of Cross-Validation and Training Accuracy

The composers discussed in Chapters 4 and 7 never computed cross-validation or training accuracy. PLOrk students did sometimes compute these measures, though they employed direct evaluation significantly more frequently. Students commonly accepted a high training or cross-validation accuracy as evidence that a model was performing well, and at least one student used cross-validation to validate his own model-building ability.

In the K-Bow work, cross-validation was used to quickly and objectively compare alternative classifier algorithms on the same dataset. This was done either after direct evaluation had shown that a reasonably good model had been built from the current training dataset, and it was uncertain which algorithm might perform most accurately, or when direct evaluation had revealed the learning problem to be particularly stubborn, and many different algorithms and feature selections were tried in succession to see if any one might result in a usable model. Cross-validation was convenient for this purpose because it provided a faster and more consistent way of comparing models than direct evaluation.

8.4.3 Correlation of Subjective Evaluation and Cross-Validation

The K-Bow study was the only study in which we collected data to compare the user's evaluation of each model against its cross-validation accuracy. For each of the bow gesture classification tasks for which three or more training iterations were performed, we computed the correlation between the cellist's subjective rating of the model and

its cross-validation accuracy on the data (computed offline, after the study). In four of the six tasks, the correlation between user rating and accuracy was negative (between -0.74 and -0.44). In these cases, some iterations' training sets contained mislabeled data and other problems such that the training set represented the learning problem in a way that was both inaccurate and too simple, leading the trained model to learn the wrong concept very well. When the cellist fixed the data sets to remove these problems, her rating of the model increased, but its cross-validation accuracy decreased. In the other two classification tasks, there was a moderately positive (0.65) or strongly positive (0.93) correlation between cross-validation accuracy and user rating, indicating that the training set likely did not mischaracterize the learning problem, and that the cellists' subjective evaluation criteria were correlated with the model's estimated generalization performance.

8.5 Model Evaluation and Generalization in Interactive Machine Learning

8.5.1 The Roles of Model Evaluation

In Section 8.4.1 above, we discussed how evaluation helped to inform users' subsequent interactions with the system. For example, identifying particular errors using direct evaluation led users to add corrective examples to the training dataset, and discovering that one learning algorithm had better cross-validation accuracy than another led to choosing the more accurate algorithm. Additionally, evaluation played an important role in training users to be more effective at using supervised learning to accomplish their goals, enabling users to practice employing trained models effectively (and assessing the degree to which they might, through practice, be able to employ them effectively in the future), and informing and guiding users' goals for the application of supervised learning to their work.

Training Users to be Better Supervised Learning Practitioners

Cross-validation and training accuracy metrics and direct evaluation served as feedback mechanisms to the users, enabling them to discover whether or not their recent changes to the training set or algorithms had had the desired effect on the retrained models. In fact, as the Wekinator did not provide any machine learning tutorials or hints, the feedback obtained from cross-validation or direct evaluation was the only mechanism for users to learn how their actions were likely to affect the system. In this way, evaluation actions trained the users—none of whom had significant experience or instruction in machine learning—to become more effective machine learning practitioners.

Users in all four studies indicated that they had learned during their interaction with the software to provide training data that more clearly expressed their intentions. By iteratively changing the training sets or other aspects of the learning problem and evaluating the results, users developed more effective strategies for interacting with

the system. For example, they learned to provide training examples that minimized noise and maximized inter-class variance and to make their models more robust by choosing which dimensions of the input gestures to vary in the training data.

Practicing and Assessing "Practiceability"

In many of the projects discussed in this thesis, users were creating trained models that they planned to use themselves in the future. Users employed direct evaluation to assess how well they were able to use a model expressively, or how well they were able to work around its shortcomings. Furthermore, several students and composers indicated that they expected to have to practice working with the trained models in order to use them most effectively, just as they would practice a traditional musical instrument. So, users employed direct evaluation to practice using the models, and in doing so assess how useful the models might eventually become as their skill at using the models improved.

Informing Users' Goals for Machine Learning

Users also employed evaluation to continually assess how well the model met their expectations, and they often adjusted their expectations or goals for the model based on the outcome. For example, they adjusted the scope of the problem to re-balance the tradeoff between the expressive capacity of the model and the time and effort required to build it. This was frequently true for problems in which users had freedom to define the gestures and mappings in creative ways. For example, several students and composers adopted model-building strategies in which they began by building simple models, then made the learning problem more complicated or more difficult over time as they discovered what the model was capable of. Also, sometimes PLOrk students building gesture classifiers consciously chose and modified the definition of each gesture class based on which gestures were most easily differentiable by the classifier they were using.

Significantly, even the K-Bow cellist, who was more restricted in her ability to re-define the learning problem of bow gesture recognition, adjusted the scope of the problem in response to discovering how well a classifier performed. For example, when the 3-class Bow Speed classifier performed very well, she decided to attempt to build a 5-class Speed classifier instead, and when the 2-class ("Up" versus "Down") Bow Direction classifier performed poorly on live bow gestures, she decided to add a third class ("Not Moving") and apply that class as a label to ambiguous examples.

As machine learning novices, users often did not have well-formed expectations of how different algorithms worked or what could be accomplished with supervised learning. They therefore also sometimes adjusted their goals when they discovered that their efforts were failing to produce a model that worked how they wanted (for example, failing to create a neural network with a smoothly linear mapping between a gesture and a synthesis parameter).

Through hands-on experimentation with the system, users also sometimes discovered that models performed in unexpected ways that they liked. This was es-

pecially true for composers and students building continuous gesture-to-sound mappings, where users had great freedom over the nature of the models to build and had the opportunity to be surprised by sounds not in the training set. In fact, many PLOrk students and composers intentionally relied on direct evaluation of the trained models to discover the sounds and gestures that they would later use in performance, rather than using the Wekinator to build a mapping between particular gestures and sounds they already had in mind.

Model evaluation also sometimes caused users to reflect on their on their model-building technique and goals in useful ways. For example, the K-Bow cellist gained a new perspective on her own bowing technique when she discovered through consistently poor system behavior that her training data was not as clear as she thought it had been. After adjusting her technique, she was able to both train a model that performed better and produce a better cello sound. One composer in Chapter 7 reflected on the quality of an instrument he had tried to build with the Wekinator, and having failed to produce a mapping he liked, he wrote, "it may be that the fact that it wanted an explicit mapping was a sign that it wasn't a very good instrument." Based on that assessment, he planned to cut it from his piece.

8.5.2 Generalization and Interactive Supervised Learning

Supervised learning algorithms are often explicitly designed with the goal of maximizing generalization performance, and standard algorithm evaluation metrics such as cross-validation typically reflect this goal. Generalization accuracy was indeed a goal for many of our observed users, in that they intended the system to produce reasonable outputs for gestural inputs that were not identical to those in the training set. In systems employing classifiers, it was important that the classifiers accurately classify gestures that were similar to the training examples. In systems employing neural networks, generalization typically played a lesser role, but it was still important to some users that gestures similar to those in the training set caused the trained network to produce sounds similar to the sounds paired with those gestures in the training data.

In the music performance scenarios for which users were creating the models, there would be inevitable variation in future inputs due to human error and environmental factors, including for example room lighting and camera position when using the video input, or non-identical sensor calibrations when using the sensor bow. To some degree, so long as model training does not take too long, a human user can mitigate the effects of a changing environment by adding additional training examples and retraining. But so long as the goal is to create a robust system for use in performance, the ability to accurately generalize to previously unseen inputs remains important.

That is not to say, though, that metrics such as test set accuracy or cross-validation accuracy remain good ways to evaluate generalization accuracy. First, users were concerned with evaluating more than the overall accuracy or error rate. Direct evaluation allowed them to identify where and how the model was likely to make mistakes, enabling them to make a cost-sensitive assessment based on criteria like error severity and the extent to which it might be possible to avoid using error-prone gestures

during a performance. Second, cross-validation is a metric designed to produce a good estimate of generalization accuracy under the condition that a fixed amount of data is available. However, in applications such as those explored here, generating additional evaluation data (i.e., gesturing during direct evaluation) is easy. Third, cross-validation estimates generalization performance by repeatedly using a held-out portion of the training set as a proxy for future, unseen data. In the case that the person driving the interactive machine learning process will also be the future user of the trained model, this proxy may be unnecessary if the user knows how to generate evaluation data that is best representative of future inputs. Furthermore, and most interestingly, in this interactive approach to machine learning, the user manipulates the training set as a means to directly influence the trained model's behavior, for example by adding properly-labeled examples to correct mistakes in error-prone areas of the input gesture space. Given that the user is consciously employing the training set to manipulate model performance, it is not necessarily reasonable to assume that the training examples will at all resemble the examples seen by the model in the future. Speaking probabilistically, it may be too strong an assumption to say that the training examples and the future examples are sampled i.i.d.[1] from a shared underlying distribution, which is a standard assumption in the design of learning algorithms and evaluation methods. So, evaluation based on a test set partition of the user-generated training data may not be meaningful.

A held-out partition of the training set may also be a particularly poor resource for estimating generalization performance during certain stages of the interactive model creation process, especially when the human has not yet discovered problems with the training data. In four of the K-Bow gesture classification tasks, there existed negative correlations between cross-validation accuracy and subjective rating for the models produced over the course of the task. In these tasks, models with high cross-validation accuracy and a low user rating can be explained as the result of the training set providing a representation of the learning problem that was both inaccurate and too simple. In all four cases, problems with the training set were undetectable using cross-validation, whereas direct evaluation allowed the cellist to discover the problems, fix them, and ultimately create models rated "10" for each task.

It is still worth questioning whether generalization performance, while undoubtedly relevant, remains more important than other objectively measurable model properties. As we've discussed, editing the training set is often the most direct and understandable way for a user to improve a model's quality, for example by deleting noisy training examples or adding new examples of inputs that were previously misclassified. The user relies on the algorithm's attention to the training examples as the primary means of influencing its behavior after training. On the other hand, many learning algorithms are designed to produce a model with good generalization accuracy (as estimated, again, from the available training data), at the cost of good training accuracy. By down-weighting the importance of training accuracy—that is, by being willing to misclassify portions of the training dataset—the algorithm may be

[1] independent and identically distributed; i.e., each example is drawn from the same distribution, and examples are mutually independent.

ignoring an intentional attempt by the user to shape the model's behavior. It is possible, therefore, that training accuracy may play a more important role in interactive supervised learning than in conventional supervised learning. While several common supervised learning algorithms such as AdaBoost, support vector machines, and decision trees privilege expected generalization accuracy through various methods, this is not universally the case. The kNN algorithm, for example, does not explicitly address generalization accuracy, and it can trivially achieve perfect training accuracy. It is possible that such algorithms may be more appropriate for certain interactive machine learning applications, and we believe that further work investigating users' algorithm preferences in interactive contexts is needed.

Finally, users' evaluation criteria reveal that the best model for a problem might not necessarily be the model with the best accuracy. Other model characteristics that users judged important were the shape of decision boundaries, the shape of the posterior probability distribution over labels, the degree to which errors were clustered in avoidable regions of the input gesture space, and the complexity of the learned function. These criteria differed among users performing different tasks. This suggests that the evaluation of supervised learning systems constructed for interactive use, such as real-time gesture recognition or audio analysis, should attend to users' goals and priorities in the application context. That is, whether or not a supervised learning system was constructed interactively or by the end user of that system, evaluating the quality of that system must take users' experiences and expectations into account; a mere use of cross-validation or other objective metrics to assess the accuracy of the trained models in insufficient.

8.6 Training Set Size and Interactive Machine Learning

Significantly, the supervised learning models built by users for which we have logging data employed training sets whose size is relatively small compared to datasets used in conventional machine learning. The average PLOrk student dataset size was 651 examples, and the average K-Bow gesture dataset included 2104 examples. The largest dataset ever used in our logged data included 11,435 examples. Discussions with composers studied in Chapters 4 and 7 suggest that their datasets were also likely small. At the same time, the majority of these users were satisfied with the models that they built, indicating that these datasets were still big enough to be useful.

This observation is significant for two reasons: First, for most learning algorithms, the computational complexity (and therefore time) of training grows as the training set size grows, so small training set sizes enable fast training. Fast training is important to the type of interactive machine learning employed by users studied here: fast training allows users to move seamlessly from evaluating models to modifying models' algorithms or data, to evaluating models again, without a disruption in interaction caused by waiting for the models to retrain. This is not to say that interactive ma-

chine learning could not be effective under circumstances demanding a longer training time of several minutes, hours or longer, but it is possible that the interface and interaction needs of end users may be different in interactive machine learning scenarios in which there is a much higher penalty incurred for each retraining action.

Second, this observation demonstrates that several key problems of interest to musicians can be addressed effectively using small datasets. This finding is of practical importance, as the small size of the datasets allows users to spend relatively little time and effort creating training examples. It is possible, again, that users would require different user interface or interaction support to effectively apply interactive supervised learning problems requiring a greater number of training examples or incurring greater time and effort in the creation of each example.

An important question, therefore, is why are so few examples necessary? Are the supervised learning problems in computer music just inherently "easier" or "simpler" than those in other domains, in which tens or hundreds of thousands of training examples (if not more) are routinely used? This could be the case, but there is another possible explanation: that the small number of training examples is possible because the systems built in this work were built with human interaction.

The fact that successful learning can take place with relatively few training examples in the applications considered in this thesis indicates that the *sample complexity* of these learning problems is low. In the PAC-learning framework (Valiant 1984; see also Section 2.2.1), sample complexity is a measure of the training set size required to claim that, with some high probability, a classifier trained on the training set and achieving a high training accuracy will in fact also have a high generalization accuracy (Eisenberg 1992). The sample complexity of a learning problem, and by extension the lower bounds on the number of training examples required to produce a probably-accurate model, are contingent on the the complexity of the concept to be learned (called the "concept class," and characterizable by the VC dimension; see Blumer et al. 1986). In other words, more complex learning problems require more training examples.

In the applications studied in this thesis, users often employed editing of the training set to interactively control the complexity of the learning problem (i.e., the concept class). For example, some PLOrk students chose a vocabulary of control gestures based on what they observed was easily learnable by the algorithm they had chosen. Even in the K-Bow work, where the bow-gesture classification problems were tightly constrained according to musical convention, the user made adjustments to the problems (such as the number of classes to use) based on her observations of classifiers' performance. In reducing the complexity of the concept class, it is possible that users were also sometimes reducing the sample complexity, making it easier to produce working models from datasets that were small enough to be created conveniently and that allowed fast training time.

The number of examples needed to create an accurate model also rests on the process used to generate the training examples. In the PAC-learning framework, the assumption is typically made that the training examples are sampled randomly from an underlying probability distribution over the space of possible examples (Blumer et al. 1986; Valiant 1984). This may be a fair assumption for many supervised learning

problems, for example when medical researchers take care to obtain data from human subjects representative of the broader population of patients, or when data mining is done on all data generated by a company's customers within a given period of time. However, this is not at all the case for the training datasets created by the interactive supervised learning users studied in this thesis. Not only do users often carefully construct the training set according to certain criteria (e.g., by providing gestures that are prototypical representations of their class), but users also, over time, learn to supply examples that allow the learning algorithm to produce a better model (e.g., by reducing noise and other non-essential variations as much as possible).

We can understand the potential for interactive supervised learning to reduce sample complexity by considering theoretical work in active learning. Active learning algorithms are suitable for problems in which there exist many unlabeled training examples, and in which users are able to provide labels for unlabeled examples when prompted. By prompting the user to label carefully-chosen examples, an active learning algorithm may need a smaller training set to construct a model with high accuracy, compared to a "passive" algorithm (such as those considered in this work) modeling the same problem. Recent work by Balcan et al. (2010) has shown that active learning can "essentially always achieve asymptotically superior sample complexity compared to passive learning when the VC dimension is finite" (where the VC dimension describes a measure of the "capacity" of a learning algorithm to model a wide set of concepts; see Burges 1998 and Vapnik 1999).

Active learning is not particularly well-suited to the type of learning applications studied in this thesis: for gesture classification and gesture mapping problems, there typically exists no unlabeled training set, and it may be just as much work for a user to create an unlabeled set as to create a labeled one. Furthermore, it may be difficult for a user to appropriately label an unlabeled feature vector (e.g., to assign a gesture label to a vector of gesture sensor outputs). However, in the interactive machine learning process studied here, the user himself plays a role similar to that of an active learning algorithm, in that he chooses additional examples to add to the training set based on their likelihood of improving the model's performance. Of course, users' assessments of this likelihood may not be accurate in any formal sense, and their choice of examples may be driven by intuition and other non-quantifiable, non-"optimal" processes. However, our observations do show that, through their interaction with the system over time, users learn something about what makes certain training examples more useful than others in building working, accurate models, and users consciously employ this knowledge when creating new training examples. In this way, it is reasonable to hypothesize that interactive machine learning might enable a lower sample complexity in practice, compared to a conventional application of supervised learning in which the training examples are generated by some non-interactive random process.

8.7 Algorithm Analysis and Design for Interactive Supervised Learning

In the work discussed in this thesis, users were able to successfully apply off-the-shelf supervised learning algorithms in an interactive manner to gestural analysis and control problems in computer music. In this section, we first outline the characteristics of these algorithms that were key to users being able to interact effectively with them in the ways discussed above. We also highlight challenges to effective interaction that these algorithms presented to users and discuss how future work might produce algorithms that afford new, useful interactions or present fewer barriers to usability. In the next section, we will discuss the properties of the Wekinator user interface that were key to enabling users to interact effectively with the algorithms, as well as control and understand other aspects of the supervised learning problem. Subsequently, we discuss how future work at the user interface level might enable more effective control or feedback for interactive supervised learning in similar problem domains.

In this section and the next, we will use the term *affordance* to refer to the qualities of algorithms and interfaces that made it possible for users to take actions that were useful to them. This term is often used in HCI in discussing the ways in which an object—e.g., a software program, a user interface element, a chair—can be used by a human actor. (Norman 1988, p. 9) describes affordances as "fundamental properties that determine just how the thing could possibly be used. A chair affords ('is for') support and, therefore, affords sitting. A chair can also be carried."

8.7.1 Algorithm Characteristics that Enabled Effective Interactions

Several characteristics of the supervised learning algorithms employed in this thesis were necessary in allowing users to employ the interactive strategies described in Section 8.2. Each of these relates to fundamental affordances of the learning algorithms, as well as the way these affordances intersected with characteristics of the concepts being learned.

Low Training Time

In nearly all applications studied in this thesis, the training time of algorithms was short enough to allow users to retrain the algorithms multiple times without experiencing an interruption in interaction. When users did experience longer training times (especially longer than a few minutes), they became frustrated and sought to take action to reduce the training time. As discussed in the previous section, the training time of most algorithms is contingent on the number of training examples, as well as the number of features and, often, subtler aspects of the relationship between inputs and outputs in the training set.

One of the algorithms used in this work, kNN, offers a unique interactive benefit in that it is an instance-based algorithm with no separate training stage. This means

that users do not have to wait for the algorithm to retrain after adding data or changing its parameters.

Low Number of Training Examples Needed

As discussed above, sample complexity is formally a function of the learning concept. Additionally, though, the number of examples needed to build a working model in practice may vary according to the learning algorithm used (Eisenberg 1992). It was important that, given the sample complexity of the concepts of interest to users, the algorithms be capable of building useful models with a relatively small number of training examples (i.e., in the range of a few thousand or less). Requiring users to provide tens or hundreds of thousands of examples to produce usable models may have made supervised learning less useful, especially without the presence of additional interface support to facilitate more efficient construction of training examples.

Appropriate Capacity of Learning Algorithm

In all applications, it was also crucial that the learning algorithm employed be capable of producing a model function that met users' needs. Generally, a given supervised learning algorithm is not capable of producing any arbitrary modeling function, regardless of how the training set and algorithm parameters are configured. Each learning algorithm has the capacity to produce certain types of functions and not others. Users became frustrated or had to change their goals when a learning algorithm did not have the capacity to learn the concept they had in mind; for example, several PLOrk students encountered problems trying to use AdaBoost boosting on decision stumps to create multi-class models. In general, the necessary capacity of a learning algorithm is likely to vary greatly from task to task.

Ability to Produce Complex Functions (Neural Networks)

A few users attempted to employ neural networks to produce mappings with specific behaviors that matched functions that they had in mind, for example to produce linear relationships between gesture features and sound parameters. This was often a difficult or impossible task, considering the set of functions producible by neural networks using a small number of nodes and a small number of training examples. This mismatch between users' goals and the algorithm's abilities resulted in neural networks not being very useful or usable to users performing these tasks.

On the other hand, most users did not employ neural networks for this purpose. It was more important to them that the model produce a function that "interesting" or "musically useful." While it was sometimes important that the algorithm have the capacity of producing a model that behaved predictably for gestures like those in the training set (i.e., a model with good training accuracy), the "capacity" of the algorithm seemed to be less of a concern than in gesture classification tasks. Instead, it was important to users that the neural networks were capable of producing complex functions in which each input parameter affected each output parameter in non-linear and interdependent ways. This affordance of neural networks was key to

easily producing models that some users said "felt" like acoustic musical instruments, and to producing models that were qualitatively different to the models users could easily create using explicit mapping strategies.

Ability to Use the Training Set to Steer Model Behavior

By definition, supervised learning algorithms produce models whose behavior is influenced by the type and amount of data in the training dataset. Wekinator users exploited this fact by using the training set to explicitly steer models toward behaviors that they judged to be desirable. Through their interaction with the training set, users were often able to iteratively refine models both to be more accurate and to exhibit a variety of properties not explicitly optimized by the learning algorithms themselves, such as musical expressiveness and low error cost.

Ability to Use Algorithm Parameters to Steer Model Behavior

The nature of an algorithm's parameters controls the range and dimensions of variations that are possible to effect in a trained models' behavior for a given learning algorithm and training set. The ability to employ algorithm parameters to influence the behavior of the trained models was occasionally useful to users in our studies. For example, the k parameter of kNN was used to control the smoothness of decision boundaries in the K-Bow gesture classification work.

Certain algorithms also had parameters that could be used to modify the amount of time an algorithm required to train. In particular, the number of AdaBoost's boosting rounds could be manipulated to incur a shorter training time, but potentially at the cost of decreased accuracy.

Fast Running Time

The algorithms employed in this work produced models capable of running very quickly: that is, very little time was needed for the model to compute an output value from an input feature vector. This was crucial to enabling real-time evaluation of the trained models, as well as in making the models usable for the real-time musical applications of interest to users. Furthermore, both training time and running time influence the feasibility of computing cross-validation accuracy, which involves repeated retraining and running of a model. If either training or running time is high, computing cross-validation can take a long time.

8.7.2 Barriers to Effective Interaction and Possible Solutions

Certain characteristics of the algorithms presented barriers to effective interactions, in that algorithms did not afford control along dimensions that were meaningful to users, or that they did not expose this control in ways that were easily usable. These barriers might be addressed through selecting or designing algorithms with additional affordances or with affordances that are more easily exercisable.

Long Training Time, Insufficient Capacity, and Insufficient Parameters

Sometimes, the extent of the above affordances was insufficient to enable users to accomplish their initial goals for using supervised learning. When confronted with neural networks that took a long time to train using the given training data and features, users sometimes attempted to shorten the training time by using smaller data sets or fewer features. This process sometimes required users to modify their ideas for how the models would work. Other times, users put up with the long training times required to accomplish their goals, but found the long training frustrating.

As mentioned above, users sometimes tried to employ algorithms to learn concepts that were not learnable by those algorithms. Sometimes the mismatch in an algorithm's capacity and a users' goals was solvable by using a different algorithm within the Wekinator (e.g., boosting on decision trees instead of stumps). Other times, another algorithm not currently supported in the Wekinator would suffice (e.g., linear regression could be used instead of neural networks to create continuous linear functions). When a user desired to build a model for which no learning algorithm would suffice, or when the user was unaware that a different available learning algorithm would meet his needs, the user was forced to modify his goals for how the system would work.

Neither modifying the training data nor modifying the algorithm parameters necessarily offered users a means to improve the model against certain of the evaluation criteria described in Section 8.4.1. Some algorithms include parameters that can be manipulated in ways to influence certain of these criteria—for example, adjusting k in kNN can adjust decision boundary smoothness—but this is not always the case. For example, decision trees and SVMs do not expose parameters that will necessarily affect boundary smoothness, and none of the algorithms offered parameters that could be manipulated to achieve particular effects on shape of the model posterior distribution.

These observations suggest that the user experience might be improved by offering a wider variety of learning algorithms—for example, to increase the likelihood that some available algorithm will meet the user's needs in terms of training time, capacity, and parameters. Additionally, learning algorithms might be designed to particularly match the constraints and goals of interactive supervised learning users in a given context. For example, algorithms could be designed with parameters for explicitly controlling boundary smoothness and function complexity, or they could be designed to allow users to specify absolute time limits on training. Additionally, engineering approaches could address high training time in the same ways that it is often addressed in conventional machine learning, for example using parallel processing on multiple machines or multiple processor cores, and/or assigning training to be done on more powerful machines or in the cloud (Armbrust et al. 2010).

Unusable Parameters

Some algorithms offered users several parameters to control the model behavior, but it was unclear to users whether or how they could use these parameters to improve

the models in ways that were meaningful and useful to them. For example, SVMs have a "complexity constant" parameter, which affects the degree to which training errors are penalized in the creation of the model. Additionally, a user must choose a kernel type (e.g., linear, polynomial, radial basis function are the kernels available in the Wekinator), and each kernel has its own parameters to set. It is unclear how changing these parameters will affect the trained model in ways that are meaningful to a user, including but not limited to the generalization accuracy of the model. Users of neural networks were faced with a similar problem: it was not clear to them how changing the network architecture, learning rate, momentum, or number of epochs would affect the model function, the time required to train the network, or any other characteristics that users cared about. Furthermore, setting these parameters to certain values sometimes resulted in errors being encountered during training, which was upsetting to users. Additionally, our conversations with users revealed that they often felt daunted by the sheer number of parameters of SVMs and neural networks, as well as by their uninformative names. Across all four studies, nobody that we know of ever changed parameter values for an SVM. While a few composers played with changing neural network architecture and parameters very early in their work in Chapter 4, nobody continued to do so.

To some degree, a user interface can be designed to repackage control over standard algorithms' parameters in a way that makes it clear to a user how the setting of a parameter will adjust a model according to his goals, as discussed in prior work by Morris et al. (2008) and Fiebrink, Cook, and Trueman (2009). However, if the available parameters are themselves insufficient to adjust a model along the dimensions of interest to a user, no user interface will be able to compensate for the barrier this presents to an algorithm's usability.

Notably, machine learning practitioners employing SVMs or neural networks in conventional contexts must themselves grapple with the problems of how to choose the SVM kernel and parameters or design the neural network architecture. While there exist conventional mechanisms for handling this (e.g., a grid search for SVM parameter values on a portion of the available data), addressing these questions effectively still takes time and effort. Practitioners must weigh this usability cost against the perceived benefits of using these algorithms as opposed to others, such as better classification accuracy or running time.

Constraints on the Learning Problem

All learning algorithms considered in this work produced a model that output a single output value in response to a vector of input features. The algorithms' lack of ability to take temporal variations over subsequent feature vectors into account constrained the types of models that could be learned. In particular, some users proposed ideas for employing the Wekinator to recognize gesture "shapes," such as numbers drawn in the air with the K-Bow or certain hand motions in front of the webcam. The Wekinator's meta-features (Section 3.3.5) are somewhat helpful in enabling users to build these models, in that feature vectors over time can be concatenated into a larger vector and then classified. However, this approach does not elegantly handle gestures

of the same class performed at different speeds, nor does it handle the requirement to identify the point at which a gesture ends (i.e., the point at which the meta-feature vector will contain features describing the completed gesture, not the gesture mid-way through execution).

The use of learning algorithms such as hidden Markov models that are capable of modeling temporal behaviors might be more appropriate for certain learning problems of interest to users. However, such algorithms would likely require different interfaces for training data creation and would involve different tradeoffs among capacity, training time, and sample size than the algorithms studied here. Further work implementing these algorithms for use in a similar interactive context is necessary to understand the interactional affordances such algorithms offer and how to build interfaces that allow users to employ these algorithms effectively.

Additionally, the fact that the Wekinator did not used temporally-aware learning algorithms, coupled with the fact that the Wekinator did not incorporate any mechanisms for segmenting the training examples into onsets or other events, made it more difficult to create high-quality training data for applications in which the modeling problem was sensitive to fine-grained timing. The composer studied in Chapter 7 who used the Wekinator to classify laptop hit position had to build additional infrastructure managing feature segmentation, so that his training examples consisted only of features extracted immediately after a hit. Some of the cello bow gesture training data had to be manually edited in the graphical editor so that the training examples only corresponded to clear bow gestures, and not moments in time between gestures.

8.8 User Interface Affordances and Challenges

We observed several aspects of the Wekinator's user interface to be important in positively and negatively impacting users' experiences. These aspects include the manner in which the algorithms' affordances are communicated and exposed by the user interface, the ways in which the user interface enabled users to control aspects of the learning problem that were important to them, and the mechanisms by which users gained feedback about the state of the learning algorithm and model.

8.8.1 Exposing Affordances of the Learning Algorithms

Users' ability to interact with supervised learning in the ways described in Section 8.2 was contingent not only on the affordances of the algorithms themselves, but on the user interfaces of the Wekinator exposing these affordances to the user in usable ways. In particular, the Wekinator software exposed to the user the affordance to steer model behavior through modifications to the training set. This interactive affordance of learning algorithms is rendered unusable, or nearly unusable, in conventional supervised learning software, in which no user interfaces exist for interactively creating and modifying the training data.

The nature of the user interfaces through which users interacted with the learning algorithms were also key to enabling users to build interactive systems that were

useful to them. The Wekinator's interface for creating training and testing data through gestural demonstration allowed users to create a sufficient number of training examples quite quickly. It also contributed to an embodied approach to designing and evaluating interactive systems. (We discuss the importance of embodiment in the next chapter.) The playalong interface allowed some users to create training examples that they felt better represented the learning problem. (And, for at least one composer, the inability of the playalong interface to smooth between sound parameter sets interfered with his ability to create better training examples.)

8.8.2 Enabling Effective Control

Much of our discussion in Section 8.2 focused on how users exercised control over the learning problem through modifying the training set, and changing algorithms and their parameters. Additionally, as discussed in Section 8.2.4, it was important for users to be able to enforce independencies between inputs and outputs through the use of feature selection. Based on our observations, even some users who had never heard the term "feature selection" found it natural to want to limit which input features affected which output parameters. This finding is understandable in light of the work by Jacob et al. (1994) studying how users perceive the integrality and separability of input devices and control tasks. Therefore, in situations in which interactive machine learning is used to build control systems where perceptually separable phenomenon are likely to be involved, feature selection should be seen not as an "advanced" machine learning technique, but as a system capability that should be easily accessible by all users.

The user interfaces used for providing control over how the training dataset was constructed also significantly impacted the usability of the system. As mentioned earlier, some users found it important to be able to design the dataset with attention to how their gestures affected one parameter at a time. Creating the Wekinator's parameter checkboxes to enable the selection of training examples for a subset of parameter models allowed users to employ this strategy. Additionally, users' ability to restrict which output parameters were influenced by certain training examples was contingent on the Wekinator software's use of an independent model for each parameter. Had multidimensional models been used (e.g., neural networks with multiple output nodes), separate training sets could not be used for each output parameter, and each training example would necessarily impact the behavior of all parameters.

8.8.3 Providing Effective Feedback

In Section 8.5.1, we discussed how direct evaluation provided a feedback mechanism that enabled users to assess the quality of the trained models, as well as become better machine learning users and refine their goals for the system. In the current version of the Wekinator, direct evaluation was the only means for providing feedback about the efficacy of users' interactions with the system, or about the feasibility of accomplishing their current goals using a given dataset or algorithm. Although it was surprisingly effective in providing these types of feedback, we do not claim that direct

evaluation is the best mechanism for this feedback, or that it should be the only such mechanism. As discussed above, a significant source of problems for users was an inability to understand how certain actions, such as changing a learning algorithm or its parameters, would affect the trained models. Additional feedback about these actions might affect the model might complement feedback from direct evaluation, and provide more efficient mechanisms that enabled users to understand the consequences of actions without having to explore the trained model. Providing additional feedback about the state of the learning system—e.g., indicating that training examples of different classes overlap in the feature space, or that the algorithm has a low training accuracy—could also aid users in identifying errors more quickly or in fixing errors more efficiently.

8.9 An HCI Perspective on Algorithms

The analysis and design of supervised learning algorithms is conventionally focused on the computational power of the algorithms, for example focusing on an algorithm's capacity to learn certain types of functions, its ability to generalize well from the available training data, and its ability to build the model efficiently (i.e., with reasonable bounds on computational complexity). These properties of algorithms are all important in interactive contexts as well, as we have discussed above. However, in order to analyze the consequences of using a given algorithm in an interactive scenario, and to examine how future work might design more usable algorithms and interfaces for interactive machine learning, we have framed the previous discussion of both interfaces and algorithms using a perspective grounded in HCI rather than in machine learning theory. In particular, we have focused on describing the affordances of algorithms that were important to users' work, and how these affordances were exposed and made usable by the Wekinator's user interfaces.

We will now outline our perspective on how the interactional affordances of algorithms shape their usefulness and usability, both independently of the user interface, and as mediated by the user interface. We will briefly analyze the algorithms used in this work based on their characteristics that contributed to or impeded users' success. Finally, we will end the section with a review of future work in algorithm design that is motivated by an HCI perspective on supervised learning algorithms in interactive contexts.

8.9.1 Usefulness, Usability, and Algorithm Affordances

As proposed by McGrenere and Ho (2000), following the original definition by Gibson (1979), an affordance is a property of an object that "exists relative to the action capabilities of a particular actor," exists "independent[ly] of the actor's ability to perceive it," and "does not change as the needs and goals of the actor change." The existence of affordances is related to the *usefulness* of an object. However, an affordance exists independently of the actor's ability to perceive its existence or the actor's ability to use it effectively (McGrenere and Ho 2000). Good design,

therefore, is concerned both with making an object useful by designing to include the affordances needed by users, and with making an object *usable* by making affordances clearly apparent to users and easy for users to exercise. McGrenere and Ho write that making affordances usable entails "account[ing] for various attributes of the end users, including their cultural conventions and level of expertise."

The question of designing a useful interactive machine learning system can be seen first as a question of choosing underlying supervised learning algorithms possessing the necessary affordances. The primary affordances needed by users in this work were described in Section 8.7.1. Some of these affordances—in particular, the capacity of an algorithm to model a particular concept, and the use of parameters to tune a model's behavior—are of routine concern to the analysis and design of learning algorithms within conventional machine learning contexts. Others—in particular, the ways in which interaction with the training set afforded users the ability to iteratively refine models to best meet their subjective goals—depart from the affordances defined in a conventional machine learning context. However, affordances are defined as the ways in which an object may be used by an actor, regardless of the intentions of the designer; in this work, users took advantage of the affordance to steer model behavior through modifications to the training set, even though this behavior departs from the conventional usage of these algorithms as typically presented to machine learning students and practitioners.

Of course, the affordances available to users are mediated by the user interface and software through which users interact with the algorithms themselves. For example, the Wekinator software did not afford users the ability to change the distance metric used by kNN, even though the Weka API exposed this affordance to the Wekinator software. We intentionally hid many algorithm parameters of this sort, in an attempt to make the software interface more usable at the expense of removing functionality that we did not judge to be too useful. However, had the software prevented users from editing the training dataset, this important affordance would have effectively disappeared, and the usefulness of the algorithms would be reduced. Had the software provided an inappropriate interface for the creation and editing of training data—for example, only an interface in which users had to manually type in every feature and parameter value for each training example—the usability of this aspect of the software would be reduced to render this affordance nearly unusable. A primary goal of building the Wekinator has therefore been to make supervised learning more useful and more usable compared to other supervised learning software, through exposing more affordances of algorithms within usable interfaces.

The design of the user interface plays an important role in informing the user of the existence of these affordances, the possibilities they present for working with the system, and the means to exercise them. A significant challenge that we encountered in this work was how to educate users, especially those who were machine learning novices, about affordances offered by the algorithms. For example, PLOrk students who encountered problems applying boosting on decision stumps to multi-class problems could have easily fixed those problems by choosing a different learning algorithm whose capacity afforded the ability to model the concept the student had in mind. However, the Wekinator interface provided no way for a user to understand this, so

some users were unable to build the multi-class classifiers they had envisioned. On the other hand, users did not seem to require extra information beyond what the Wekinator user interface offered them in order to understand how to employ the training set effectively to control the behavior of the trained models.

While user interfaces may improve the usability of supervised learning algorithms by exposing their affordances in understandable and usable ways, we argue that a user interface cannot add new affordances to a learning algorithm, and that usability of a machine learning system can be fundamentally limited by qualities of the underlying algorithms. For example, a user cannot employ kNN for regression, no matter what the user interface looks like. A user cannot apply a decision stump to create an accurate model for a three-class classification problem. Furthermore, kNN's parameters can be tuned to make the decision boundary smoother, but no matter what the user interface to kNN looks like, the algorithm's parameters cannot be guaranteed to be tunable to enforce a linear decision boundary (even though a linear hyperplane is within the set of decision functions capable of being produced by kNN for certain training sets). A decision tree's parameters, on the other hand, can be tuned to enforce a linear boundary (i.e., by limiting the tree to one level).

In summary, the affordances of algorithms themselves determine the potential usefulness of the system. A user interface can make an algorithm less useful by hiding affordances, or it can provide access to the affordances that are important to users (even affordances not typically exploited in conventional machine learning). Certain algorithms are more or less amenable to embedding in usable interface, in that the affordances they expose may be more or less difficult to present to users in a meaningful way, and for users to learn to use effectively. While a good user interface may make an algorithm easier or more difficult to use, the usability of the system remains fundamentally limited by the affordances exposed by the algorithm, as the user interface cannot compensate for an algorithm that does not afford control over aspects of the model or learning process that are important to users.

8.9.2 Improving Algorithms' Usability and Usefulness in Interactive Supervised Learning

Potential improvements of interactive supervised learning systems can therefore take three somewhat different directions: the design of algorithms with new and more useful affordances, the design or modification of algorithms to make their affordances more usable, and the design of interfaces that make algorithm affordances more apparent and easier to exercise effectively.

In this work, we have focused on improving usefulness and usability by exposing more of algorithms' innate affordances to the user—particular, the ability to create and modify training data, and to evaluate on new examples—and on providing usable interfaces for exercising these affordances.

Prior work by other researchers has focused on adding new affordances to the underlying algorithms, including work in both HCI focused on adding affordances that take advantage of human interaction in new ways (e.g., work by Shilman et al.

2006 and Kapoor et al. 2010), and work in machine learning research. We next outline a few other possible avenues for designing new algorithms for interactive learning by creating new affordances and making them more usable.

One significant example of machine learning algorithms being developed to take advantage of human interaction is the development of active learning algorithms, which we described above in Section 8.6. Active learning algorithms leverage human interaction in domains where much unlabeled data is available and where human interactors are capable of providing labels to unlabeled examples selected by an algorithm. Active learning algorithms can improve the usability of supervised learning in these domains, because humans may have to spend less time and effort labeling data that may not be useful to the algorithm. One could imagine a similar type of interactive learning algorithm that prompts the user not for labels, but for feature vectors. In the musical problems considered here, a natural part of many users' working process was to come up with gestures that produced sounds they wanted to be able to control. By extension, it could be easy for a user to listen to a sound generated by a set of target outputs selected by a learning algorithm, then to demonstrate one or more gestures (i.e., input feature vectors) to add to the training set along with the parameters producing that sound. (In practice, for continuous mapping problems, it would also be useful for a user to be able to indicate that she did not want that sound to be part of the training set, or producible by the trained model.) Such a learning algorithm could select its prompts with the goal of improving usability by making the training set creation process more efficient, or with the goal of improving classifier accuracy by selecting prompts from classes likely to be misclassified.

Another avenue for algorithm development might be to explicitly design algorithms whose parameters enable tuning of the models along dimensions that are important to users. For example, algorithms might be designed to include parameters that allow a user to explicitly adjust decision boundary smoothness, function complexity, or the tradeoff between training time and model accuracy. To some degree, a user interface can be designed to repackage control over standard algorithms' parameters in a way that makes it clear to a user how the setting of a parameter will adjust a model according to his goals, as discussed in prior work by Morris et al. (2008) and Fiebrink, Cook, and Trueman (2009). However, it may not always be straightforward or even possible to re-parameterize an existing algorithm to provide control against the dimensions of interest to a user, so new algorithms or new variations on existing algorithms might be useful. Additionally, algorithms could be designed to better take advantage of a user's knowledge of the real-world costs of misclassifications, as was done for example in work by Kapoor et al. (2010). To some degree, this might be addressed through allowing the user to interactively apply weights to classes or examples, communicating their relative importance to the learning algorithm. In Weka, most standard classifiers can take advantage of weighted instances, so in future work we plan to augment the Wekinator user interface with this ability.

In addition to allowing users to interact with algorithms in new ways and use parameters more effectively, algorithms might be designed to enable users to exercise control via the training sets more effectively. As we discussed in Section 8.5.2, this might be accomplished through the use of algorithms that prioritize training accuracy

over generalization, or through algorithms which expose a regularization parameter to the user.

8.10 Significance of Domain Characteristics in Interactive Machine Learning

Based on our work presented in this thesis as well as prior research in interactive and conventional machine learning, there are several characteristics of a problem domain that we see as important in establishing whether interactive machine learning is feasible in that domain, in defining the types of interactions that are possible, and in shaping what users require from a system's algorithms and interfaces. These characteristics include the extent of users' expertise in the application domain, whether the interactive machine learning user will also be the end user of the trained models, whether the application domain involves real-time interaction, and the extent to which users may be flexible in adapting their goals for the supervised learning process. For each of these criteria, we describe why it is relevant to our results and how our findings in this thesis intersect with existing or potential interactive machine learning applications or research in other domains.

8.10.1 What is the User's Expertise?

In applications studied in this thesis, users were familiar with the learning problem (i.e., the concept the model would learn) and proficient in the domain (e.g., producing hand gestures or cello bowing articulations). These characteristics were essential in making it appropriate and feasible for users to create training examples and to create the new inputs used to directly evaluate models. Therefore, our findings regarding how users developed strategies for creating and modifying training data, and for how they evaluated the trained models by generating new inputs, are most relevant to other domains where this is the case. For example, this type of interaction might be useful to enable domain experts in medicine, the sciences, or other technical domains to build models whose goal is to replicate—but not augment or surpass—the expertise of a trained human. Additionally, there are many pursuits in which many humans are "experts," including understanding speech, perceiving objects visually, driving a car, etc. In creating computer systems capable of such pursuits, programming by demonstration has long been used to address computational challenges by leveraging human expertise.

8.10.2 Who Uses the Trained Models?

In many of the applications studied in this thesis, users were engaged in building models that, once trained, would be used only by them. This enabled users to consider simplified versions of the learning problems: for example, the K-Bow cellist only had to create a set of classifiers usable by her, so she did not have to consider how to create training or testing data that might be representative of cellists of different

sizes or skill levels. Users also had a high degree of knowledge about the types of applications in which the models would be employed, and about the sorts of future inputs the models would see. This further simplified the learning problem, and it also meant that users could evaluate trained models by providing inputs that were similar to those likely to be used in practice. Other domains in which end users routinely assist the computer in building models or control vocabularies to be used only by them include speech recognition and gestural gaming systems.

Furthermore, for some of the applications studied here, users had some degree of control over how they would use the models in the future. By gaining experience with the models, they could identify where models were likely to fail and potentially avoid using failure-producing inputs in future interactions with the system. The extent to which this is the case in other applications, such as speech recognition and gaming gesture recognition, is dependent on the application.

In applications for which the interactive machine learning user is not the end user of the trained models, the interactive machine learning user must find a way to accommodate the increased uncertainty and variability in future model inputs. A conventional machine learning approach that uses a fixed dataset for training and/or evaluation may be appropriate, or the user might augment data generated by other users with his own. Collaborative approaches in which multiple users interactively contribute training data or evaluate trained models could be useful, for example in the construction of general-purpose speech or gesture recognition systems.

8.10.3 Real-time or Offline?

In all of the applications studied here, users created models that would ultimately be used to produce real-time outputs in response to a stream of real-time inputs. The real-time nature of the problem domain presents unique challenges: for example, it is difficult to imagine a way to produce concise visualizations of the problem space or trained model, as was done in work on web image classification by Amershi et al. (2009). On the other hand, the real-time nature of the domain—coupled with the domain expertise of users—lends itself well to creating interfaces for users to generate training examples and evaluation examples in real-time. Users were able to generate training examples very quickly by demonstration, and they were able to evaluate the models in real-time in a manner very similar to how they planned to use the final set of trained models in the future.

The direct evaluation mechanism of the Wekinator allowed users to employ an exploratory approach to evaluation of the trained models. They were free to quickly assess the model behavior over a wide range of the input space, focus on how the model function varied with small and subtle changes in inputs, or focus attention only on the parts of the model space that they had just edited. We did not make a detailed assessment of users' strategies for exploring the models during evaluation, but the actions that people took in order to improve the models (discussed in Section 8.2) do underscore that the evaluation strategies that people developed were informative of a variety of evaluation criteria. An interesting avenue for future work might be to investigate whether rapid, open-ended exploration of the trained model space

(within a user interface for conducting this exploration efficiently) is beneficial for users applying supervised learning to offline domains, such as image classification. Using a gestural controller and a mapping from the controller into the model space, users could even employ gesture-driven exploration of models in other domains, which might allow for both rapid exploration and an embodied approach to understanding the modeling problem. On the other hand, future research could also investigate how to extend approaches for feedback in offline domains (e.g., by Amershi et al. 2009) into real-time domains. For example, instantaneous visual feedback on qualities of the model function that are important to users could allow users to focus their direct evaluation on exploring the most informative areas of the model space.

8.10.4 How Flexible are Users' Goals?

In the applications studied here, the degree of control that users exercised over the scope and definition of the learning problem ranged from completely open-ended (e.g., designing both the gestural vocabulary and its discrete or continuous mapping to sound) to quite constrained (i.e., designing K-Bow models that classified gestures according to conventional string technique). In open-ended applications, users were free to change nearly every aspect of the learning problem in order to create a model that worked in a particular compositional context. For example, they were free to make the learning problem easier by using fewer classes or by using different gestures when they found it difficult to train a model according to their original plans, or they were free to take advantage of surprising behaviors of the mapping function. On the other hand, the K-Bow user was less flexible in her ability to make the learning problem easier, and some PLOrk students were intent on creating particular types of gesture recognizers.

Based on our observations, we believe it is likely that the extent to which users' goals are flexible influences the nature of the support they require from the algorithms and user interface. For example, a user who must create an accurate model of a very specific concept may have a greater incentive to compare many algorithms, parameter settings, and feature selections before choosing the best of them; a user who must create an accurate model of a less specific concept may be able to modify the concept to be easier to classify, and he may be able to do so more easily than finding a more accurate learning algorithm. Highly-constrained users may therefore require a wider assortment of algorithmic tools (including learning algorithms, feature extractors, etc.) as well as greater interface support in helping them to understand these tools and wield them effectively. Less-constrained users, on the other hand, might require tools to help them efficiently identify ways that the learning problem might be re-defined to produce better accuracy, and to explore the consequences of choosing among possible problem definitions. In our user studies, users often did not fall cleanly into one or other other of these categories, so we could not directly compare the needs of these groups. However, we believe that future work more rigorously exploring these needs of these two types of users would be informative.

Additionally, we underscore that even the most constrained of our observed users, the K-Bow cellist, was flexible with regard to certain aspects of the learning problem,

such as the number of gestural classes used and (sometimes) the precise locations of the decision boundaries. In many conventional supervised learning application domains, from gesture recognition to medical diagnosis, the translation of a real-world phenomenon into a supervised learning problem definition likely involves a set of subtle and sometimes flexible human decisions about the number of classes, what constitutes a correct or incorrect classification, and other aspects of the problem. In conventional applications, these decisions are implicit and fixed in the training set. However, our observations suggest that interactive supervised learning by domain experts can enable these flexibilities to be exploited in appropriate ways. Further research might investigate the nature of these flexibilities in a range of conventional domains and demonstrate how allowing them to be manipulated interactively by end users enables the construction of better models.

8.10.5 Other Domain Characteristics

Computer music is a domain in which systems-building can often be a long, arduous process, and one that does not always engage the creative or musical strengths of users. As we discuss in the next chapter, composers particularly valued the Wekinator's ability to enable a high-level, embodied, and efficient approach to systems design, in contrast to their experiences with other tools. Our observations suggest that interactive machine learning may be useful as a tool in domains for which supervised learning isn't always computationally necessary, but where users may benefit from a tool for high-level design and rapid prototyping. This encompasses many creative domains, as we discuss in the next chapter. Prior work by Hartmann et al. (2007) has also found interactive supervised learning to be suitable under these circumstances, in sensor-based interaction design prototyping.

Additionally, Wekinator users sometimes found that interacting with supervised learning provided a useful perspective on themselves as creators of examples or demonstrators of gestures. For example, the K-Bow cellist learned through model-building that two of her articulations were not as clearly distinct as they should have been, and she was able to improve her technique as a result. One of the case study composers (Weitekamp) and many of the PLOrk students learned to provide very clear and consistent gestures, because not doing so resulted in models that performed poorly. This suggests that interactive machine learning might be a useful tool in application domains where computers are used to provide feedback on human actions, including feedback about consistency with oneself or with an expert. Such domains might include music pedagogy (e.g., teaching someone to play an instrument), sports pedagogy and training, physical therapy, and speech therapy, among others.

8.11 Conclusions

8.11.1 Why Interaction Matters

Our observations of users applying interactive supervised learning to their work and the success with which diverse users employed the Wekinator to create a variety of useful musical systems underscore the fact that interaction can play an important role in enabling end users to employ supervised learning algorithms effectively.

Interaction with the training dataset allows users to employ their expertise in defining the nature and scope of the learning problem, to correct errors in a model, and to iteratively improve a model against their own criteria for success. Furthermore, modifications to the training set allow users to change the nature of the learning problem as their goals for the learning process change, or to take advantage of newly-acquired knowledge of what types of training data will most likely create the desired model qualities. Interactively modifying learning algorithms, their parameters, and the features they use are also sometimes important, but modifying the training data is often the most direct way of influencing the behavior of the trained model to better meet users' goals.

Interactive evaluation of the trained models allows users to assess models against their own criteria for success, which include accuracy as well as a range of other characteristics, such as decision boundary shape, complexity, misclassification cost, and posterior confidence, which are likely to be different for different users and applications. Interactive evaluation informs users how they might take action to correct shortcomings of the models by modifying the training set or other aspects of the learning problem. Interacting with trained models can also enable people to discover new capabilities of models that they hadn't expected, and it can allow people to develop and practice strategies for using the models most effectively.

Repeatedly modifying the data or other aspects of the learning problem and evaluating the effects of those modifications serves to inform users over time about the types of interactions that are most likely to produce the desired outcomes; in other words, iteration enables users to develop effective strategies for interactive supervised learning. Users also learn over time which model behaviors are easily obtained, and which are more difficult. Ultimately, users employ interaction to discover and manage tradeoffs among their goals for the learning problem and the effort required to accomplish those goals. Interaction can also be useful in helping a user to formulate and refine their goals, as we discuss in the next chapter.

8.11.2 Summary and Future Research

Our work in this thesis has demonstrated that interaction offers numerous practical benefits to end users applying supervised learning to their work. Additionally, our work has highlighted some key differences between interactive and conventional machine learning applications, particularly with regard to the role that generalization accuracy plays in interactive contexts. Future work might explicitly investigate the relationship between generalization accuracy and users' goals in a variety of domains,

and it might explore whether algorithms that build models with attention to criteria other than generalization accuracy are useful in interactive contexts.

Another open question is the extent to which human creation of the training dataset in interactive contexts serves to reduce the sample complexity of learning problems. It may be the case that aspects of the strategies naturally adopted by interactive supervised learning users are effective in reducing sample complexity, or that users can be encouraged through education or user interfaces to adopt strategies that enable them to use less data. Either case would bode well for the possibility of using interactive supervised learning to effectively and efficiently create models with few training examples in other domains.

In this chapter, we have suggested that the HCI framework for understanding affordances, usefulness, and usability can be applied to the analysis of supervised learning algorithms. We believe that qualities of algorithms intersect with qualities of the user interface in determining the usefulness and usability of the system. Understanding these qualities can inform choices of algorithms to use in interactive contexts, and future work designing new algorithms to be more useful and usable interactive contexts will be valuable. For example, new algorithms could be designed to take advantage of different human interactions, and to provide parameters that correspond to dimensions of behavior that are important and understandable to users.

We have also discussed how the user interfaces for control and feedback presented opportunities and challenges to users of the Wekinator, independently of the learning algorithms used. There exist remaining design challenges on how to support and encourage effective interaction with machine learning algorithms, particularly by users who are machine learning novices.

We have achieved a good understanding of the nature and breadth model evaluation criteria and goals employed by users in our studies. Future work might study the criteria and goals of users in other application domains, such as gesture recognition in non-musical applications. If commonalities across domains are found—for example, in users' desire for increased control over decision boundary smoothness or training time—these may suggest fruitful avenues for the design or modification of learning algorithms to be used across many different domains.

In this chapter, we have outlined characteristics of the computer music application space that have been crucial to enabling the application of interactive supervised learning to users' work. Our findings generalize most clearly to domains that share similar characteristics: where users are domain experts, constructing models for themselves to use, and working in real-time domains; extension of interactive machine learning to different types of domains presents interesting research challenges. One domain characteristic which we believe has a significant impact on users' requirements, but which has been unexplored in past research, is the degree to which users are able to redefine aspects of the learning problem. In future research, we hope to explore how interaction may allow users to exploit areas of flexibility not only in creative work, but also in more conventional application domains.

Chapter 9

Discussion: Interactive Supervised Learning and Creativity

9.1 Introduction

As we discussed in Section 2.2.3, prior research has demonstrated that supervised learning can be a useful tool in several key problems in computer music composition and performance, including the creation of gesture-to-sound mappings, the classification of human gesture, and the semantic analysis of audio. In this thesis, we have demonstrated that it is both feasible and valuable to enable end users to take an *interactive* approach to applying supervised learning to their work in gesture analysis and mapping.

Some of our goals for the Wekinator outlined in Chapter 2 included the creation of a supervised learning tool that worked in real-time, that was compatible with a range of tools that composers used, and that was possible to use without programming. Clearly, the Wekinator software meets these goals, and these characteristics have been instrumental in allowing the range of musical projects discussed in this work. The software has been used successfully by professional composers and undergraduate students, and by people building expressive musical instruments and gesture classifiers. Musicians have used it to design interactive systems controlled by a range of input modalities and controller hardware, and to control sound and visuals within the ChucK, Max/MSP, and Processing environments. Users in every study indicated that they were able to use the Wekinator to do their work more efficiently, to create systems with great potential for musical expressivity, and to build new instruments that they would not have been able to create using other methods. Furthermore, users were able to accomplish these things without, in most cases, knowing much at all about machine learning.

Based on these observations, we claim that interactive supervised learning is an appropriate and useful tool for creative users working in computer music. In this chapter, we more deeply interrogate why this is so: How has interactive supervised learning functioned to support the creative activities of the users we have observed? In particular, we examine interactive supervised learning in the context of prior work

on creativity support tools and embodied cognition, and we demonstrate how it serves several important creative functions.

Subsequently, we review how our observations of and work with Wekinator users contribute to a deeper understanding of the human-computer interaction requirements of composers. Our work both illuminates the values and goals held by users engaged in music composition and instrument building and highlights roles that technology may play in helping users achieve their goals. In particular, composers in our studies valued being able to use technology to access inspiration and to engage their musical, creative, embodied expertise in their work. Our work specifically sheds new light on the importance of interaction and algorithms in enabling generative mapping strategies to offer distinct benefits to digital musical instrument designers. Our findings also emphasize the utility of broadening the scope of discussion about interaction in computer music to include the processes of composition and instrument design, and we propose a view of the Wekinator and other compositional tools as "meta-instruments."

9.2 Interactive Supervised Learning and Creativity Support

9.2.1 Creativity Support and HCI

In Section 2.5 in Chapter 2, we briefly outlined the existing thread of research in HCI on the use of technology to support human creativity. Fundamental research in this area, particularly work by Shneiderman (2000, 2007), is aimed at understanding people's creative processes and the opportunities for technology to support types of thinking and working that are common across creative domains. Also relevant is a growing body of domain-specific work investigating the needs of particular user communities and assessing the usability of software designed for those communities.

In 2005, a group of seven creativity support researchers—people familiar in both foundational and domain-specific research on HCI and creativity—proposed a set of design principles for tools to support creative thinking (Resnick et al. 2005). Their intention was for these principles to "guide the development of new creativity support tools—that is, tools that enable people to express themselves creatively and to develop as creative thinkers." This set of principles was amalgamated and refined from prior research and previously-proposed design guidelines, and like much of that prior research, the authors define creative work broadly. Creative work encompasses not only work in the arts and other "creative" domains, but work such as software engineering, architecture, science, and education in which people musts "be not only more productive, but more innovative."

In particular, the design guidelines proposed by Resnick et al. pertain to what the authors call "composition tools," where "composition" is used not in the musical sense, but in the sense of "computational systems and environments that people can use to generate, modify, interact and play with, and/or share both logical and/or physical representations. A creative composition process is not a routine production

process that can be prescribed, and what tools and representations people use strongly affect their courses of actions and thought processes."

The design guidelines for composition tools proposed by Resnick et al. can be paraphrased as follows:

1. **Support Exploration:** A tool should allow users to try many alternatives, explore the space of possibilities, backtrack when actions are unsuccessful, make it clear to users how the tool might be used, and support "sketching" of incomplete ideas.

2. **Provide a Low Threshold, High Ceiling, and Wide Walls:** A tool should be usable by novices and experts, and it should support and suggest a diversity of applications.

3. **Support Many Paths and Many Styles:** A tool should support different learning styles and approaches and even work against traditional biases toward certain cognitive styles.

4. **Support Collaboration:** A tool should allow each person in a team to contribute according to their strengths and to work in parallel.

5. **Support Open Interchange:** A tool should seamlessly operate with other tools used in creative work, and it should be extensible (e.g., providing support for user-generated plug-ins).

6. **Make It As Simple As Possible, And Maybe Even Simpler:** The user experience should be simple, even for tools that allow users to accomplish complex tasks.

7. **Choose Black Boxes Carefully:** Black boxes or "primitive elements" determine what ideas users can explore and what ideas remain hidden. The choice about what to hide or reveal should consider the goals and knowledge of the target audience.

8. **Invent Things That You Would Want to Use Yourself:** Tools should be enjoyable, and they should foster communities of users who enjoy using them.

9. **Balance User Suggestions With Observation and Participatory Processes:** Observing how users work can be more informative than asking them for suggestions; participatory methods can involve substantial effort but lead to projects that are better accepted by the broader community of users.

10. **Iterate, Iterate—Then Iterate Again:** Tool designers should constantly revise their tools, using design prototypes with potential users when possible.

11. **Design for Designers:** Design tools that enable users to design, create, invent, explore, and reflect; foster creative thinking, not a "paint-by-numbers" approach to creative work.

12. **Consider the Evaluation of Tools:** "It is still an open question how to measure the extent to which a tool fosters creative thinking," but longitudinal studies over long periods of time can lead to deep insights about whether a tool is helpful and why.

Of these guidelines, numbers 1–8 and 11 pertain to qualities of the creativity support tools themselves, as well as principles for designers to consider when creating them. Guidelines 9, 10, and 12 refer only to guidance for the design process. In this chapter, we focus on the former set of guidelines in our discussion of the usefulness of interactive machine learning as a creativity support tool. However, we note that our own experiences working with composers in the participatory design process in Chapter 4 revealed the guidelines regarding the design and implementation process to offer sound advice. Iteration, participatory design, and longitudinal study were essential in enabling us to improve the usefulness of the Wekinator software and to understand users' needs more deeply.

9.2.2 Interactive Supervised Learning and Creativity Support Tool Design Principles

As a tool for enabling end users to create, modify, and interact and play with supervised learning systems in music, the Wekinator falls squarely into the the scope of "composition tools" targeted by the above design guidelines. Indeed, our work with users of the Wekinator reveals that many of these guidelines intersect with the priorities of users employing technology in their work, and that the manner in which the Wekinator implemented certain of these design ideas contributed significantly to its usefulness and usability. In this section, we examine many of these design principles and discuss how they were important to users and how they were supported by the interactive supervised learning process in the Wekinator.

Support Exploration

The Wekinator's support for exploration and prototyping was of crucial importance to composers. At the conclusion of the participatory design process in Chapter 4, when asked what aspect of the Wekinator was most useful to their work, four of the composers' responses included mention of the speed with which the Wekinator allowed them to create and experiment with new mappings. Composers strongly contrasted their experiences experimenting with the Wekinator with the difficulty and slowness of creating and changing mappings using other tools.

Interactive supervised learning was essential to supporting exploration and prototyping with the Wekinator. First, given a working feature extractor and synthesis algorithm, the construction of an initial prototype could be completed in a few minutes; all that was necessary was the demonstration of a few training examples and the training of a model. Because the training process completed so quickly, the time between choosing some gestures and sounds to explore and experimenting with the newly-created instrument or controller was often only a few seconds.

Within the interactive supervised learning paradigm, the actions of revising models or starting over completely were easily executable and understandable. Incremental changes could be induced by adding training examples that encoded the way in which the user wanted the model to change, and changes could be rolled back by deleting one or more recent "rounds" of training data and retraining. Starting over involved simply deleting the training set and adding new examples. Some composers commented that the ease of creating and changing mappings in the Wekinator enabled them to build instruments that they were ultimately happier with than those built using explicit mapping techniques: not only were they able to explore more possibilities, but they were less likely to feel they needed to keep a mapping they disliked simply because of the amount of time and effort that had gone into building it.

Because of the ease of rapidly building prototypes in this way, some composers actually proposed applying the Wekinator for building interactive systems that did not strictly require machine learning. For example, one composer in Chapter 4 proposed using the Wekinator to construct a text command interface in which feature vectors were character buffers and class values were the identities of commands. Even though machine learning is not at all necessary to recognize text commands, the effort required to write the feature extraction code and provide some command examples may be less than that required to construct a standalone, real-time command interface for controlling a synthesis patch. Furthermore, using machine learning to implement such an interface allows additional capabilities, such as some robustness to misspellings and the ability to define new commands on-the-fly.

Resnick et al. also discuss the usefulness of sketching in creative exploration, and users in our work frequently employed the training dataset as a sketching tool. Both the K-Bow cellist and PLOrk students using the Wekinator to build classifiers sometimes began by sketching out simpler versions of the classification problem, for example using only a subset of the classes of interest and a few training examples. This enabled them to build a first prototype even more quickly than using more and more carefully-chosen examples, and it enabled them to discover almost immediately whether their current approach to building the classifier (i.e., their chosen features and/or input device coupled with their chosen gestures and classifier) was capable of working.

Composers constructing continuous mappings often used the training dataset to sketch out the boundaries of the sonic and physical gestures they were interested in using in the mapping, and they relied on the neural networks to "fill in" this sketch with detail. Because the neural networks were used as a continuous mapping function, they were able to literally fill in the modeling function in between the training examples, so that users of the trained models could employ any gesture—even gestures very different from the training examples—and still produce sound. Neural networks were thus usable for turning a rough sketch of a couple examples into an immediately-usable, fully-functional instrument. Composers could then incrementally add detail to the sketch by adding more training examples then expect the neural network to fill in around these new details after retraining.

In addition to saving composers time by filling in the details of the sketch, the continuous nature of the neural networks meant that the trained models were capable of producing output values not in the training set. When the networks controlled a sound synthesis algorithm, this meant that composers could use the trained models to discover new sounds that they hadn't employed in the training data, and which they might not have even imagined previously. In their gesture-driven explorations of the trained models, they could also discover new gesture-sound relationships that they hadn't anticipated. Both composers and PLOrk students emphasized the creative value of discovering new sounds, both as a way to inspire further development of the composition or instrument currently being built, and to inspire future compositions. Furthermore, once new sounds or gesture-sound pairs had been discovered, interactive machine learning made it easy for users to take advantage of these discoveries. By adding them to the training set as additional examples, users could often effectively ensure that they would remain accessible in future versions of the mapping.

The neural network algorithms themselves were useful in filling in composers' sketches in a musically constructive way, in that they facilitated the creation of complex mappings. First, the use of several neural networks in parallel, each of which computed an output as a function of many or all input features, enabled an easy way to construct many-to-many mappings. Furthermore, neural networks are capable of creating highly nonlinear, mathematically complex functions. The use of many-to-many relationships (Hunt and Wanderley 2002) and complex functions are musically valuable: acoustic musical instruments are characterized by such "functions" from performer gesture to perceived sound, and users of the Wekinator often commented that the mappings they built "felt" more like acoustic instruments. However, these properties can be difficult to design efficiently and effectively in an explicit mapping, where considerable effort must be spent considering the functional role of each input in relation to each output.

Low Threshold, High Ceiling, and Wide Walls

Interactive supervised learning offered a "low threshold" to accommodate users who were machine learning novices, as well as people who were new to instrument design and to programming. Users who did not understand how the machine learning algorithms worked still found it easy to grasp how to influence the nature of the trained models by making modifications to the training set. Furthermore, the feedback they received through model evaluation allowed many users to learn to improve their strategies for providing useful training data, for example by minimizing noise or choosing classes that were easily distinguishable in the feature space. Users who were new to instrument design, computer music, or gestural interfaces were also able to use the Wekinator successfully, both because the Wekinator was packaged with built-in feature extractors and example synthesis patches, and because the use of interactive machine learning accommodated users who did not understand the nature of the features being extracted or how the synthesis algorithm parameters were used to produce sound. Instead, users only had to manipulate the synthesis parameters to find some initial sounds they wanted to use, then record training examples by demon-

strating the corresponding gestures. Furthermore, the interaction with the machine learning algorithms could be performed effectively within a GUI, so users neither had to program nor understand machine learning well enough to use a machine learning API or library.

Interactive supervised learning can also offer a "high ceiling" for users who are knowledgeable in machine learning, programming, and/or composition and instrument design. Although none of our users discussed in this thesis were machine learning experts, someone who is knowledgable about the way particular algorithms construct models from the data or how algorithm parameters are likely to affect the models can exploit this information in interacting with the system. The modular nature of supervised learning means that users can design highly-customized feature extractors, learning algorithms, and/or synthesis algorithms or other processes without having to modify other aspects of the system or change the way in which they interact with the system. (In fact, in work with the Wekinator not discussed in this thesis, a user who is a researcher proficient in machine learning and programming implemented a new learning algorithm class to be used within the software.) Furthermore, as our observations of users show, interactive machine learning is a useful tool for users who are musical experts. By leveraging their musical expertise in the process of both evaluating models and improving models using new training examples, the Wekinator allowed these users to create instruments that were both musically expressive and highly customized to their compositions. These users also leveraged the modularity of machine learning, in that they used the outputs of standard learning algorithms to control their own expertly-refined software for sound synthesis and music performance.

The generality of machine learning algorithms further contributes to fulfilling the "wide walls" principle: because supervised learning algorithms are designed to effectively create models from data without regard to the application domain, users are free to apply them to connect virtually any type of inputs to virtually any type of outputs. The diversity of projects created by users observed in this thesis attests to the width of the Wekinator's walls. Users' proposals for other projects that might be accomplished with new feature extractors and controllable software processes are even more diverse, spanning real-time audio analysis, interactive animations, and gesturally-controlled games.

Support Many Paths and Many Styles

In terms of approaches to creating models and mapping functions, interactive machine learning clearly privileges a high-level, holistic approach to systems-building, as opposed to a low-level approach characterized by applying logical reasoning to the construction of the model function. Certain interactions with the Wekinator were, however, structured to allowed people who preferred to think in lower-level ways to work more efficiently. In particular, feature selection enabled users to efficiently communicate their ideas for how certain input dimensions controlled certain output dimensions, and training example selection enabled them to engage a more piece-wise approach to model creation, in which they considered the effects of each training ex-

ample on just one (or a subset) of the sound parameters. Nevertheless, when users had very specific mathematical ideas about how they wanted the models to work—for example, to create a linear function of inputs to outputs, or to trigger an event when a feature value surpassed a specific threshold—users indicated that they would prefer to explicitly design these functions in code rather than use the Wekinator.

On the other hand, some composers did not like feature or example selection at all, and all composers valued the fact that the Wekinator allowed them the opportunity to think and design at a high level. Other tools that composers had used—in particular, using programming to explicitly create instrument mappings—strictly enforced a low-level, mathematically-based approach to constructing instruments and compositions. Those tools were not only less efficient to use in constructing the types of many-to-many, complex mappings that composers valued; they also enforced a way of thinking about the design and composition process that was counterproductive and did not engage composers' musical expertise. One composer in Chapter 4 wrote, "I basically want to retrain myself NOT to think that way anymore, and rather to privilege physical interactions with the interface and sonic sculpting on the synthesis end."

Interactive supervised learning supports a high-level, holistic approach to design in that the models are created from training examples, and in creating the training examples, users can simultaneously communicate information about the way that many dimensions of input features will influence many dimensions of model outputs. Users can design from the top down by specifying behaviors they want the model to have; the learning algorithm does the work of translating the prescribed behaviors into low-level functions capable of producing those behaviors. Additionally, debugging and improving systems can also be done at this high level, as users can interact with the trained model to identify behaviors that they would like to change, and then specify additional training examples to communicate the new behaviors.

A related and significant benefit of applying interactive machine learning to work in music is that, by providing interfaces for creating training and evaluation data through real-time demonstration, the high-level process of design, evaluation, and improvement also leverages musicians' *embodied* expertise and knowledge. We further discuss the importance of embodiment in creative work in music composition in Section 9.2.3.

Resnick et al. propose that one component of the "many paths" principle is to work against traditional biases toward certain cognitive styles. Although interactive machine learning does not effectively support all ways of thinking and doing, it does provide a set of approaches to thinking about and implementing design that are not well-supported by existing tools. By providing an alternative to programming and programming-like interfaces in which low-level aspects of the design must be reasoned about individually, interactive machine learning can be a useful tool for creative work by people who are averse to thinking in these ways, for creating projects that can be more effectively implemented and improved without dissection in these ways, and for work by people who do not know how to program. In our work, it was more important to create a software application that broadened the types of ways people could approach the design process, and the types of people who could participate, than to try to accommodate ways of working and thinking that are already well-supported

by existing tools. Interactive machine learning was fundamental to creating such an application.

Support Open Interchange

Compatibility with many other software and hardware tools was key to making the Wekinator useful to the students and composers with whom we worked. At the interface level, we established this compatibility through the use of OSC communication and a ChucK API (see Sections 3.3.2 and 3.3.4), which allowed the Wekinator to obtain features from any type of feature extractor and send its outputs to control any type of dynamic process that conformed to its simple communication protocol. OSC even allowed the support of "coordination across windows," a specific suggestion made by Resnick et al., by synchronizing synthesis parameter states between the Wekinator and external synthesis programs (see Section 3.3.4).

Underneath the user interface, the general-purpose, modular nature of supervised learning algorithms was also valuable to supporting compatibility. As mentioned above, the learning algorithms are naive to the nature of the input features and output values, so users can apply these algorithms to highly-customized problems in many domains.

Choose Black Boxes Carefully

The modularity and generality of supervised learning algorithms also lend themselves to encapsulation in appropriate "black boxes." In order to apply an algorithm to a given problem, a user does not necessarily have to understand the inner workings of the algorithm, only the fact that it requires a training dataset and a training action in order to produce a model that computes outputs from inputs. Opening up the algorithm black box can be useful; the more a user understands about how the algorithm works and how its parameters function, the more information she has about how she might apply it more effectively. In the Wekinator, therefore, we did not completely hide control over the algorithm parameters so that users who needed to manipulate them could do so. However, our observations show that many creative tasks in music may be accomplished effectively without understanding or manipulating the inner workings of the algorithms.

The fact that users could interact with the training set was particularly important to enabling algorithms to be treated as black boxes. In conventional machine learning, when the goal is to create a model for a fixed dataset, a greater proportion of the options available to a user for improving a model are contingent on his understanding something about the algorithm. However, when a user has the ability to create and modify the dataset itself, this allows him to engage a set of strategies for modifying model behavior (discussed in the previous chapter) whose effects are, to a great extent, independent of the choice of the underlying algorithm. Furthermore, the feedback that users receive about their actions through the evaluation of the trained models enables them to develop practical strategies for more effective interaction while leaving the algorithm in its black box.

Importantly, interactive machine learning is able to support creative work in these ways because only the algorithm is in a black box—the training set and the supervised learning process itself are not. These properties are central in distinguishing the creative possibilities available with interactive machine learning from those available in conventional, non-interactive machine learning.

Design for Designers

Composers using the Wekinator highly valued its potential for providing them with access to surprise and inspiration. The Wekinator was not just a tool for them to efficiently build mappings they had in mind; it was a tool for presenting them with inspiring new possibilities and facilitating the creation of new instruments and systems that embodied these possibilities. The ways in which users could interact with machine learning and the properties of the algorithms themselves that have been described above were all key in supporting composers in the creation of novel, expressive, musical systems.

At a high level, the usability and usefulness of the Wekinator rested on composers' ability to rapidly iterate between using models—to evaluate them, practice them, and explore them—and modifying the training data or other aspects of the system, including the algorithms, controllers, and synthesis algorithms. This iteration enabled composers to leverage their musical skills in the design process to create systems that met their goals, as well as to develop and evolve their goals as they used the Wekinator to discover new sonic and gestural possibilities. The fact that the Wekinator not only allowed them to easily change and refine their goals (e.g., through modifications to the training data), and the fact that it seemed to actively suggest certain musical possibilities and even defy absolute control by the composer, were fundamental to its creative utility.

One composer in Chapter 4 summed up his experience with the Wekinator: "The Wekinator enables you to focus on what your primary sonic and physical concerns are, and takes away the need to address so many details, and it does so in such a way that even if you DID spend all the time on building the mappings manually, you would *never* come up with what the Wekinator comes up with. So, the process becomes more focused, more musical, more creative, more playful. I actually *want* to do it."

9.2.3 The Importance of Embodiment in "Performative Creativity"

Although embodiment is not discussed in the design guidelines by Resnick et al., our observations and work with users suggest that embodiment plays an important role in musical creative practice, and it should be emphasized in tools intended to support creativity in musical and other "performative" contexts involving human gesture and real-time motion.

In cognitive science, *embodiment* refers to "the role of an agent's own body in its everyday, situated cognition" (Gibbs 2006, 1). Whereas Cartesian mind/body

dualism considers learning, knowledge, and reasoning to be mental phenomena that take place independently of the body, understanding cognition as an embodied phenomenon presents ways of recognizing these activities as arising from a deep integration of the mind and body together.

Klemmer et al. (2006) have summarized the value that an understanding of "human embodied engagement in the world" offers to the design and evaluation of interactive systems. Two themes they emphasize are *thinking through doing*, which "describes how thought (mind) and action (body) are deeply integrated and how they co-produce learning and reasoning," and *performance*, which "describes the rich actions our bodies are capable of, and how physical action can be both faster and more nuanced than symbolic cognition." Considering these themes provides an additional set of of perspectives on why supporting embodied thinking and interaction can be an asset to creativity support tools, and to creative tools for music in particular.

In discussing "thinking through doing," Klemmer et al. raise several points relevant to our discussion: (1) "Humans learn about the world and its properties by interacting within it." (2) Epistemic actions, which involve "manipulating artifacts to better understand [a] task's context ... are one of many helpful ways in which a user's environment may be appropriated to facilitate mental work." (3) People think by prototyping. "Reflective practice, the framing and evaluation of a design challenge by working it through, rather than just thinking it through, points out that physical action and cognition are interconnected ... The epistemic production of concrete prototypes provides the crucial element of surprise, unexpected realizations that the designer could not have arrived at without producing a concrete manifestation of her ideas."

These points suggest that systems used for designing interaction, exploration, learning, and creation—all activities targeted by the "composition tools" discussed by Resnick et al.–can leverage a broader range of human skill and leverage this skill more effectively by considering the embodied role of the human actor. In the Wekinator, these skills were leveraged by engaging a user's body in the creation of training data and in the evaluation and exploration of models. Not only that, but the gestural input device, trained model, and synthesis algorithm together can be seen as constituting a prototype instrument—an embodied, concrete artifact in itself.

In discussing "performance," Klemmer et al. raise several additional relevant points: (1) "Physical tacit knowledge is an important part of professional skill." (2) "We are able to sense, store and recall our own muscular effort, body position and movement to build skill. It is this motor, or kinesthetic, memory that is involved in knowing how to ride a bicycle, how to swim, how to improvise on the piano." (3) "[S]peed of execution ... favors bodily skill for a class of interactive systems that require tight integration of a human performer 'in the loop.'" (4) "One of the most powerful human capabilities relevant to designers is the intimate incorporation of an artifact into bodily practice to the point where people perceive that artifact as an extension of themselves; they act through it rather than on it."

Given the importance of exploration and reflection in the creative process, these points suggest that, when creative tools are used to build systems for physical interaction, such tools should engage users' physical knowledge, kinesthetic memory, and

capabilities for tightly-integrated control in evaluating, discovering, and even realizing the creative artifacts. In engaging the user's body in designing and evaluating models, the Wekinator enabled composers to employ their physical knowledge gained through years of musical training. For some users, especially the K-Bow cellist, this tacit knowledge directly involved the knowledge of how to play a specific musical instrument. In the design of instruments that used new physical gestures, users were able to employ the musical skill of performing physical actions while listening critically to the sonic response, all the while reflecting on their own technique and the actions required produce useful sounds. In other words, users were able to employ an *enactive* approach to gestural-sonic exploration, the utility of which has also been noted by Wessel (2006) and Armstrong (2006) in their work on digital instrument design.

Furthermore, the hands-on evaluation of models was used by composers not just to assess the ability to which they could employ the models expressively, but to practice using the models and assess the extent to which, after further practice, they might be able to employ them more expressively in the future. That is, composers relied on their kinesthetic memory and kinesthetic understanding to evaluate and later use the models.

Musical instruments clearly fall into the category of systems that require tight integration of humans "in the loop" and bodily skill, and the goal of many composers using the Wekinator was arguably to build new instruments that—like acoustic instruments—became extensions of the performers rather than interfaces that were "acted on." Users were able to employ hands-on evaluation in the Wekinator to assess whether the instruments they had created were capable of this type of expressive playability.

Composers themselves often discussed the significance of having access to a software tool that enabled them to compose in an embodied way. For example, one of the composers studied in Chapter 7 (Trueman) writes: "With [the Wekinator], it's possible to create physical sound spaces where the connections between body and sound are the driving force behind the instrument design, and they *feel* right. It's very difficult to do this with explicit mapping for any situation greater than 2–3 features/parameters, and most of the time we want more than 2–3 features/parameters, otherwise it feels too obvious and predictable. So, it's very difficult to create instruments that feel embodied with explicit mapping strategies, while the whole approach of [the Wekinator], especially with playalong, is precisely to create instruments that feel embodied."

9.3 Understanding and Supporting HCI in Computer Music Composition

9.3.1 Technology and Composers' Priorities

Our work observing composers using the Wekinator and engaging them in the process of improving and evaluating the software has led to an understanding that several

interactive affordances offered by the Wekinator are of critical importance in the composition process. In Chapter 4, we identified key priorities that were important to composers in their interactions with technology in the composition and instrument design process. Those priorities include: the speed and ease with which they could create and explore mappings, privileging the gesture-sound relationship via physicality and abstraction, access to surprise and discovery, access to complexity, the ability to balance surprise and discovery with predictability and control, and an invitation to play. Many of these priorities dovetail with the general creativity support guidelines proposed by Resnick et al. and others. While physicality and embodied interaction are not discussed by Resnick et al., these were among the most significant values held by composers, and they pertained to the characteristics of the Wekinator that most distinguished it from other compositional tools. The composers who participated in the case studies in Chapter 7 and the PLOrk students in Chapter 5 echoed that many of these priorities were relevant both to continuous mapping and discrete gesture classification tasks.

The K-Bow cellist applied the Wekinator in an appreciably different scenario. Because her goal for the models was to accurately classify standard cello bow gestures, she was less free to innovate in the design of the models themselves, and she was not as interested in being surprised by the behaviors of the trained models or in creating models that were unnecessarily complex. However, her work with the Wekinator was still embedded within a larger creative context, in that she was interested in using the models for controlling sound and visual systems in live performance. First, she appreciated that, by creating accurate models of her cello gestures, the Wekinator enabled her to build more natural performance-time interactions into her compositions: "It allows me to augment the bowing skills I spent years working on." She also valued that interacting with the trained models during the model creation process enabled her to assess their suitability for use in performance—e.g., she could discover whether they made mistakes on gestures she was likely to use in a piece—as well as help her to imagine ways to use them creatively. In particular, she liked the richness of the information available in the models' posterior distribution outputs, which suggested the use of model certainty or uncertainty in driving both feedback mechanisms to a user as well as continuous parameters of the computer performance program. She also valued that she could modify the models in ways that might make them easier to use in imagined performance scenarios—for example by adding extra classes to a classification problem to indicate that a gesture was "None of the Above" and so should be ignored. Additionally, she found it useful that she had the option to quickly re-create or modify models before a performance to adjust for changes in sensor calibrations on the K-Bow itself.

9.3.2 Interactive Supervised Learning and Instrument Design

As discussed in Section 2.2.3, prior research has proposed a dichotomy of generative and explicit techniques for creating mappings in new musical interfaces. In Section

4.5.2, we proposed that many of the qualities of the Wekinator that were most useful to composers arose from its nature as a generative mapping tool. We now revisit that idea in the context of the other user studies and the above discussion on creativity.

The use of supervised learning algorithms to construct mapping functions is relevant to the support of many of the creative design principles discussed above. In particular, the ability to build mappings from examples as opposed to designing the mapping functions in code allowed for the efficient construction of complex, many-to-many mappings, and it made the instrument-building process more accessible to non-programmers. The general-purpose and modular nature of the functions enabled applicability to a wide range of input devices and control tasks. However, the interactive aspect of the supervised learning process was also fundamentally important to realizing the creative benefits of generative mapping: it was only through the ability to interactively create the training data, interactively evaluate the trained models, and iteratively modify the mappings that users were able to take advantage of the full benefits of designing, exploring, and refining at a high conceptual level and in an embodied manner. In other words, the creative potential of the Wekinator arose both from the fact that data-driven algorithms were used to create the mapping functions and the fact that the interactive affordances (see Section 8.7) of these algorithms were exposed to the user in an appropriate, usable interface.

Additionally, certain characteristics of the learning algorithms themselves may have been important to supporting creative work in music. The neural networks used to create continuous mappings afforded users the ability to create complex, non-linear functions that—when coupled with complex physical modeling synthesis algorithms—reminded them of acoustic instruments, required practice in order to be played well, and exhibited behaviors that surprised and inspired users. Furthermore, the training time of these algorithms was usually short enough that they afforded efficient, frequent mode-switching between mapping evaluation and mapping modification.

Neural networks are not the only learning algorithm capable of exhibiting these characteristics. A broader class of useful mapping-building algorithms might be defined as those offering the affordances most needed by instrument-building users. In particular, this class of algorithms could be characterized by the following affordances: providing a means of creating a many-to-many function from some training data (either through multiple parallel models or a single multidimensional model), creating that function quickly, and possibly creating a function that is complex (e.g., incorporating non-linearities, discontinuities, or other behaviors). Many algorithms in this family might not be considered to be "supervised learning" algorithms in any traditional sense, and indeed, dynamic time warping and other algorithms that have been used for mapping creation fit into this family although they are not supervised learning algorithms per se. Defining this algorithm family based on the fit of the algorithms' affordances to users' needs, rather than the computational approaches used to build the model, gives us a framework for analyzing a given algorithm's suitability for use in mapping. It also suggests a set of design criteria for creating new mapping-creation algorithms that does away with the pretense of supporting generalization accuracy or otherwise functioning like a machine learning algorithm.

Our observations of and conversations with users suggest several new such functions that might be embedded into future versions of the Wekinator. For example, algorithms that inject randomness into the mapping function could enable users to explore several alternative ways of "filling in" a training data sketch for the same training examples. Such algorithms could expose parameters for users to control the dimensions and extent of random variation. Also, algorithms that designate "islands" of model flatness in portions of the feature space surrounding the training examples could enable users to reach the sounds present in the training set more reliably by giving them a larger gestural "target" area for sounds they have indicated are important to them.

Finally, our work with composers and students has shown that some users, including both expert and novice instrument designers, do sometimes find that explicit mapping strategies are useful or necessary. Sometimes users imagine aspects of an instrument working in very simple ways, for example a single synthesis parameter value varying linearly with a single gestural input dimension. In these cases, users may desire to explicitly specify and manipulate the mapping function at a low level. This behavior is not well-supported in the current version of the Wekinator. However, adding support for mapping-creation algorithms that do afford lower-level manipulation of the model functions could address this issue. For example, embedding linear and polynomial regression algorithms into the Wekinator could enable users to still take advantage of the high-level design, embodiment, and other benefits of a data-driven mapping creation paradigm, while also allowing them the option to directly edit the mapping functions after they are created from the data.

9.3.3 Interaction in Music Composition

Much prior work studying and theorizing about human-computer interaction in computer music has focused on performance-time interactions. For example, the work by Hunt and Kirk (2000) described in Section 2.1.2 studied the consequences of mapping function properties on real-time musical control. Composers' perspectives on musical human-computer interaction discussed in Section 2.1.2 also largely pertained to the role of the human, the computer, and the control and influence between the two during live performance.

A theme across several of these composers' perspectives is concern that the definition of human-computer interaction—and, by extension, the implementation of interactive possibilities in musical systems—not be limited to a one-way exercising of control by the human over the computer. Instead, they valued a definition of interaction and the creation of musical experiences that included the capacity for non-determinism, complexity, and rich mutual influence between human and computer.

Many composers observed in our work valued these same characteristics in their interactions with the computer in the process of composition and instrument design. The ways the software challenged and influenced them were important aspects of their experience as users, and many users routinely relied on influence by the computer to shape the gestural vocabularies they constructed, the sound worlds of their new in-

struments, the aesthetic nature of their compositions, and the roles of the instrument and human in performance.

These observations suggest that the scope of discussion surrounding human-computer interaction in computer music should be expanded to include consideration of interaction during composition and instrument building. Composition-time interaction has been considered in the writings of individual composers reflecting on their working process (e.g., Hahn and Bahn 2003), by a few HCI researchers studying composers (e.g., Tsandilas et al. 2009), and in some recent work discussing interaction from a broad, musical perspective (see Paine 2009). However, relatively little discussion has focused on the relationship between human-computer interaction in composition and performance at a general level (i.e., as opposed to within the context of a particular composition). We have found it useful to reflect on our work studying human-computer interaction in composition within the context of the larger body of work on performance-time interaction by understanding the Wekinator as a "meta-instrument," as we describe next.

Design Tools and Meta-Instruments

The description of "creativity support tools" by Shneiderman (2000), as well as the scope of subsequent research employing this term, leaves room for the two-way, mutual interaction and influence between users and tools that is so important to composers. The creativity support tool guidelines described above recognize the value of users being inspired and influenced by technology. However, the definition of software such as the Wekinator as a "creativity support tool" is somewhat problematic, in that it implies a subtle distinction between the support tool itself and the creative work being supported. Composers using the Wekinator were typically constructing instruments and compositions that could be seen as both containing and deriving their identity from the models created by the software. Also, the instruments designed using interaction with the Wekinator themselves contained the Wekinator software embedded inside of them, patching gestural inputs to sound parameters during live performance. Thus, the Wekinator was simultaneously a tool that enabled users to build an instrument, a piece of configurable software that users turned into an instrument, and a set of compositional possibilities defined by the dimensions along which it could be configured.

Dan Trueman, one of the composers studied in Chapters 4 and 7 who has used the Wekinator extensively, takes exception to using the term "creativity support tool" to describe the Wekinator. In e-mail to the author on 15 October 2010, he wrote: "I tend not to think of the software that I use as 'support tools' but rather as contexts and instruments in themselves that suggest particular kinds of musical possibilities. They aren't necessarily [a] means to an end, but rather creative spaces. [The Wekinator] is particularly rich in this regard, both when using it to actually build an instrument, and then finally when using that instrument post-training."

Since the Wekinator's creation, we have referred to it ourselves not as a "creativity support tool," but as a "meta-instrument" (Fiebrink, Trueman, and Cook 2009). As an instrument for creating instruments, the Wekinator invites the understanding that

users engaged with it are expressing themselves in real-time through their work, in a way that requires musical expertise and real-time, embodied skills. This perspective is particularly suitable given the fact that, at any one time, a user of the Wekinator may be simultaneously composing a piece, designing an instrument, practicing the instrument, and rehearsing the piece, among other activities. (In fact, in one of our first uses of the Wekinator, in a piece called *nets0*, users were engaged in all of these actions while simultaneously performing the unfolding piece in front of an audience.[1]) Each of these activities, like playing an instrument, involves the real-time translation of human intention into musical expression.

Musical Interaction with Meta-Instruments

Viewing the Wekinator and other technologies for supporting composition as meta-instruments suggests a way to reconnect our analysis of interaction with the large body of work discussing interaction in performance. For example, it is useful to considering interactions with the Wekinator through the lens of interactive performance paradigms and metaphors proposed in the computer music literature.

As a meta-instrument, the Wekinator supports many of the interactive values described by composers that we discussed in Section 2.1.2. For example, the Wekinator supports a high "potential for change in the behaviors of computer and performer in their response to each other" in accordance with to Moon (1997). The Wekinator presents an interface in which "interaction transcends control" in accordance with David Rokeby as described by Rowe et al. (1993), and it supports a relationship between user and computer akin to an ongoing, mutually-influential conversation, such as that described by Chadabe (2002).

Arguably, the mutual influence involved in instrument design in the Wekinator can facilitate the creation of more complex, surprising, and even difficult mappings than those created by explicit strategies. These properties can enable instruments created with the Wekinator to involve performance-time interactions that, while still incorporating deterministic mappings, themselves incorporate richer styles of mutual human-computer influence than those possible with simpler mappings. For example, composer Michelle Nagai indicated that, because of the complexity of the relationship between gesture and sound in instruments created by the Wekinator, the attention of the performer and audience were likely to be more focused on the experience of the piece and not on decoding the mechanisms used for control. For Dan Trueman, instruments created with the Wekinator were designed to challenge and influence their players: "[The instrument] requires practice to explore and master, and is constantly revealing new possibilities, but is fairly easy to get started on; the player doesn't need to spend lots of hours learning the instrument before joining the piece—rather the instrument teaches the player how to play it..."

[1] Video of *nets0* is available at `http://www.cs.princeton.edu/~fiebrink/thesis/resources.html`.

9.4 Conclusions

In this chapter, we have discussed the Wekinator and the style of interactive supervised learning that it supports in the context of HCI design guidelines for creativity support tools. Our work has demonstrated that interactive supervised learning can effectively support creative work through assisting in the exploration of design possibilities, rapid prototyping, and sketching; accommodating novice and expert users working on a range of tasks; supporting a high-level and holistic approach to design; integrating well with other software tools; enabling learning algorithms to be treated as black boxes when appropriate; and providing users with access to surprise, complexity, and inspiration. Furthermore, through engaging embodied interactions with the training data and trained models, interactive supervised learning can leverage embodied knowledge, skill, learning, and memory in the creative design of interactive systems.

In this chapter, we have also discussed how our work in this thesis has led to a deeper understanding of how technology can support composers in the process of composition and instrument design. Composers and students creating new instruments valued the speed and ease with which they could create and explore mappings, privileging the gesture-sound relationship via physicality and abstraction, access to surprise and discovery, access to complexity, the ability to balance surprise and discovery with predictability and control, and tools that presented an invitation to play. The K-Bow cellist valued the speed and ease with which she could construct accurate and musically-useful models of her natural cello gestures.

We have also emphasized how, due to both the affordances of particular algorithms and the interactive context, interactive supervised learning offers advantages to the instrument design process. We have proposed the utility of defining, analyzing, and designing algorithms according to the match between their affordances and the instrument design process. Finally, we have outlined our understanding of the Wekinator as a meta-instrument and discussed how it embodies the performance-time interaction values of composers, especially the possibility for rich, mutual influence between the human user and the computer.

In the future, we plan to continue working with musicians applying the Wekinator to their work to further improve it as a creative tool. For example, improving the range of algorithms supported, the tightness of integration with other tools using OSC, and the simplicity and informativeness of the user interface may make the Wekinator a more usable tool for a wider range of applications and users. In addition, we are interested in using the Wekinator to support and study creativity in other application domains. Interactive art and game design, for example, are other domains in which interactive supervised learning could serve as a creative design tool, and in which users may similarly benefit from embodied approaches to system design and evaluation.

The idea we find most exciting in this work is that interactive supervised learning systems can support embodied and high-level approaches to design, as well as provide users access to complexity and discovery. These characteristics are, we believe, hard to achieve in software systems, but they may be of critical value in supporting efficient and innovative creative work in many domains.

Chapter 10

Conclusion

10.1 Summary and Contributions

In this thesis, we have examined applied machine learning through the lens of human-computer interaction. In doing so, we have created, studied, and improved systems for users to interactively apply supervised learning algorithms to their work in computer music composition, performance, and instrument design. We have created a useful software tool that has aided students and professional composers in creating new musical works, and in doing so, we have demonstrated the feasibility and efficacy of interactive machine learning in this application domain. Our work with users has led to a clearer characterization of the requirements and goals of interactive machine learning users and of the different roles that interaction may play in allowing them to design and evaluate systems, to learn to become more effective users of machine learning, and to work creatively. As a result, this work has both empowered musicians to create new forms of art and contributed to a broader HCI perspective on machine learning practice.

In this section, we present a summary of our work and highlight the contributions that are most significant to future research in HCI and machine learning, as well as to research and creative work in computer music.

These contributions include:

1. A new software tool allowing real-time human interaction with supervised learning algorithms and, within it, a new "playalong" interaction for training data creation.

2. A demonstration of the important roles that interaction—encompassing both human-computer control and computer-human feedback—can play in the development of supervised learning systems, and a greater understanding of the differences between interactive and conventional machine learning contexts.

3. A better understanding of the requirements and challenges in the analysis and design of algorithms and interfaces for interactive supervised learning in real-time and creative problem domains.

4. A clearer characterization of composers' goals and priorities for interacting with computers in music composition and instrument design, and a demonstration that interactive supervised learning is useful in supporting composers in their work.

5. A demonstration of the usefulness of interactive supervised learning as a creativity support tool.

10.1.1 The Wekinator

Our work has produced a new software tool, the Wekinator, that allows end users to interactively apply supervised learning learning to their work in real-time problem domains. It is general-purpose in nature, in that users may apply it to creating trained models that analyze gesture, audio, or other arbitrary real-time input signals and produce outputs that drive sound synthesis, visualizations, or other arbitrary dynamic processes. It is tailored for use in music: it comes packaged with a set of audio and gesture feature extractors and example synthesis patches; and it uses Open Sound Control (OSC: Wright and Freed 1997), a communication protocol common in music and media software, to support compatibility with other software systems designed and used by composers. It supports a rich set of user interactions with the supervised learning process, including the creation of training data by real-time demonstration and the evaluation of trained models through real-time demonstration of testing examples and observation of model behaviors. It also supports interactive, iterative modification of the training data, the selection and configuration of learning algorithms, and the selection of features.

10.1.2 Playalong Learning

In addition to enabling users to create training data by demonstrating gestures that correspond to a fixed set of model outputs, we have enabled Wekinator users to create training using a "playalong" interface in which they gesture along with a changing "score" of model outputs, as if they were controlling those outputs in real-time.

Some of the composers who used the Wekinator found playalong useful in creating data that captured more fine-grained aspects of their ideas for how model outputs should vary with the input features. Additionally, composers used playalong to engage their own musical and physical expertise more deeply in the model-building process. It allowed them to use a more embodied approach to instrument design and to create models that captured something about how an instrument should "feel" in expressive, real-time performance.

10.1.3 Demonstrating the Feasibility and Usefulness of Interactive Supervised Learning in Music

We have observed and collaborated with a variety of musical users applying the Wekinator to their work. During a 10-week participatory design process with seven

composers, we made many improvements to make the software more usable and useful to composers building expressive new instruments. We used the Wekinator as a teaching tool with 22 undergraduate students, who used it to build new expressive controllers and gesture recognition systems. We also collaborated with a professional cellist/composer using a sensor-equipped bow to interactively build classifiers capable of recognizing standard cello bow gestures. Finally, we conducted interviews with three composers—an undergraduate student, a graduate student in composition, and a professional composer—who have used the Wekinator in publicly-performed computer music compositions.

The outcomes of this work with Wekinator users first underscore the feasibility and usefulness of applying interactive supervised learning to a range of applications in computer music. Most of the users studied in this work knew very little or nothing about machine learning before starting their work with the Wekinator, yet the software enabled them to apply standard supervised learning algorithms effectively to create a variety of interactive music systems. These systems included expressive new musical instruments, which used supervised learning models to translate sensed performer gestures into continuous changes in synthesis algorithm parameters; as well as systems for the recognition of standard or novel discrete performance gestures (such as cello bow articulations or physical hits to the laptop) and the translation of these gestures into changes in sound or visualizations.

Students were successful in building accurate classifiers and expressive instruments using the Wekinator in their coursework. The cellist/composer was able to build classifiers with which she was highly satisfied, and which performed comparably to or better than classifiers created by researchers using conventional (non-interactive) machine learning on the same problems. Composers indicated that the Wekinator enabled them to work more efficiently and to build new instruments that were more musically expressive than those built in other ways. Furthermore, several composers remarked that the Wekinator enabled them to attempt and succeed in creating musical systems or compositions that they would not have attempted or even imagined without the use of the software.

10.1.4 Demonstrating The Usefulness of Interaction in Supervised Learning

Our studies of these users have also led to new knowledge about the roles that interaction can play in allowing end users to successfully create supervised learning systems. Specifically, in our studies, interactive creation of the training set allowed users to define the scope of the learning problem, to communicate the essential characteristics of each class, to preemptively minimize error and cost of the trained model, and to sketch out areas of interest and denote boundaries of the input gesture and output sound spaces. The "playalong" interface for data creation allowed users greater control over the learning concept expressed in the training set and further engaged their physical and musical expertise in the design process. Interactive modification of the training set allowed users to correct errors, to take advantage of new and desirable

sounds they discovered during their work, to make models more complex, and to roll-back changes they had made. Interactive evaluation of trained models enabled users to evaluate models against their goals for the learning process; assess how they might improve models through modifications to the training data and learning concept; compare models produced by different learning algorithms; and identify problems with the training data not identifiable using cross-validation. Interactive evaluation also trained users to become better machine learning practitioners, served as a method of "practice" in which they assessed and improved their own abilities to behave in ways that produced the desired model outputs, and led to refinements and changes in users' goals for the machine learning system.

Through repeated iterations of model evaluation and modification, users were able to construct interactive systems that met their goals. Significantly, throughout these iterations, users worked to improve the performance of the trained models according to their criteria for success, while also refining their goals for how the trained models should function and how the models would ultimately be used. Users sometimes simplified the problem when models performed poorly, sometimes increased the complexity of the problem when models performed well, and sometimes changed the nature of their goals entirely when they discovered unexpected new behaviors in the models. The abilities to iteratively evaluate models against an array of subjective criteria and to change the training set to reflect their evolving goals for the system were crucial.

10.1.5 Understanding the Differences between Interactive and Conventional Machine Learning

These studies also led to a better understanding of differences between interactive and conventional supervised learning that may be useful in designing new learning algorithms for interactive applications. Importantly, generalization accuracy was sometimes relevant to users' goals for the system, but it was not the only or most important evaluation criterion that users employed. This suggests that future work might investigate whether existing or new algorithms that build models according to other criteria might be appropriate for interactive scenarios, and it suggests that evaluation by human users should supplement conventional metrics such as cross-validation accuracy in the evaluation of models intended to be used in interactive contexts.

Additionally, the size of the training sets employed by users in this work was often small compared to those used conventional supervised learning. This offers a practical advantage, in that short training times lead to less interruption between model modification and model evaluation actions. We hypothesize that the smaller size of the training sets is due to users adjusting the difficulty of the target learning concept to ensure that it is possible to learn with a training set size that is both feasible to interactively create and allows short training times, and/or to the fact that users learn through experience to provide training examples that are more effective than randomly-chosen examples in modeling the target concept.

10.1.6 Understanding Requirements and Challenges of Algorithm and Interface Analysis and Design

Through our analysis of how users interacted with supervised learning within the Wekinator, the characteristics of the software they most valued, and the challenges to effective interaction that remained following our work with composers to improve the system, we have come to understand how the interactional affordances of a learning algorithm play a role in shaping its usefulness and usability. Among the interactional affordances that were key to making standard supervised learning algorithms usable in this work were their low training time, their capability for building models for the chosen learning concepts using a small number of training examples, their fast running time, and their ability to be "steered" in different directions via users' modifications to the training set. Future work to improve algorithms for interactive contexts could focus on adding new interactive affordances, for example mechanisms similar to active learning that prompt the user to provide training examples that are anticipated to be useful. Or, future work could focus on making the algorithms more amenable to embedding in a usable user interface, for example by designing algorithms with tunable parameters that can more be easily manipulated to change the model along dimensions of interest to a user.

Characteristics of the interface that were key to making the Wekinator usable included exposing the above affordances of the learning algorithms, rather than hiding them from the user (for example, conventional machine learning interfaces hide the affordance for tuning model performance using the training set); enabling effective control over feature selection to allow users to create systems with independencies between features and outputs, and enabling training example selection to allow users to employ a training data creation strategy in which they considered one output parameter at a time; and providing feedback to the user about the state of the learning system and the efficacy of users' actions. Future work might improve the user interface to more effectively address the challenge of providing useful, timely, and fine-grained feedback over learning system state, the actions available to a user to interact with the system, the consequences of past actions, and the likely consequences of future actions. In particular, this work might explicitly address the challenge of providing this information in a manner that is suitable to machine learning novices.

The ways that users employed algorithms and interfaces, and by extension, the properties of the algorithms and interfaces that were important to them, were contingent on several characteristics of the problem domain. In the applications studied in this thesis, users were domain experts (i.e., experts at performing the gestures used to provide input to the models). The users were building models for themselves or for other users they understood well, creating models that would be used in a real-time context, and working with a set of goals that ranged from mostly inflexible to completely flexible. Future work might investigate interactive supervised learning in other domains whose characteristics vary along these dimensions to identify commonalities or differences in users' interaction and algorithm needs, thereby generalizing our findings to other domains and identifying the most fruitful avenues for improving algorithms and interfaces for a wide population of users.

10.1.7 Demonstrating the Usefulness of Interactive Supervised Learning as a Creativity Support Tool

Users' work with the Wekinator has demonstrated that interactive supervised learning can be a useful tool in creative contexts: the Wekinator enabled users to not only work more efficiently and productively than other tools they had used, but it also inspired them and helped them to create musical compositions and systems to express themselves in new ways. In fact, many of the ways that users in our studies employed the Wekinator can be understood as taking advantage of the capabilities of interactive supervised learning to function as a creativity support tool. We have discussed how interactive supervised learning effectively addresses many of the requirements of creativity support tools as proposed in work by Resnick et al. (2005) and others. In particular, our work has demonstrated that interactive supervised learning can effectively support creative work through assisting in the exploration of design possibilities, rapid prototyping, and sketching; accommodating novice and expert users working on a range of tasks; supporting a high-level and holistic approach to design; integrating well with other software tools; enabling learning algorithms to be treated as black boxes when appropriate; and providing users with access to surprise, complexity, and inspiration.

Furthermore, Wekinator users felt strongly that they benefited from the physical, embodied approach to design that was supported by the software. Through engaging embodied interactions with the training data and trained models, interactive supervised learning can leverage embodied knowledge, skill, learning, and memory in the creative design of interactive systems.

10.1.8 Understanding Composers' Priorities for Human-Computer Interaction in Composition

Finally, our work has led to a clearer characterization of composers' priorities for human-computer interaction in composition and instrument design. Composers and students creating new instruments valued the speed and ease with which they could create and explore mappings, approaches to design that privileged the gesture-sound relationship via physicality and abstraction, access to surprise and discovery, access to complexity, the ability to balance surprise and discovery with predictability and control, and tools that presented an invitation to play. Additionally, the cellist valued the speed and ease with which she could construct accurate and musically-useful models of her natural cello gestures.

Composers using the Wekinator to build new, expressive instruments especially valued the way they could use the Wekinator for inspiration, surprise, and discovery. Their work with the Wekinator was not a one-way exercising of control over technology to build a model that met their specifications; rather, it was a mutually influential, richly interactive process. By considering the Wekinator as a meta-instrument, these types of interactions can be seen as fulfilling the requirements of real-time musical interactivity put forth by composers such as Moon (1997) and Chadabe (2002) in their writings about performance-time interactions.

10.2 Future Work

10.2.1 Improvements to the Wekinator

Our discussions with composers and musicians have provided us with many ideas for further improving the Wekinator as a creativity support tool and meta-instrument. Some of these improvements involve supporting new interactions to benefit composers' work. For example, adding parameter smoothing to the playalong process will enable composers to create training datasets capable of faithfully representing a broader range of goals, and to create training data in a way that better takes advantage of composers' embodied expertise. Incorporating additional interactions for generating and editing training data, for example enabling users to annotate video of performers executing gestures and to "replay" recorded gesture sequences through the trained models, may facilitate deeper and more natural control over the training data in ways that benefit composers and musicians, as well as dancers and others interested in gestural interaction. Adding mechanisms to visualize the training dataset and decision boundaries might provide additional means to understand how the learning algorithm is working, thereby helping machine learning novices to better understand the effects of their actions and helping users to identify and fix problems when the models do not work as they expect.

Another set of improvements will focus on making the software interface even simpler and easier for more people to use. The current terminology in the interface (especially with regard to features and parameters) is confusing to novice users and should be changed or augmented with more helpful contextual information. Further integration with OSC—for example, enabling users to send OSC commands to load and save learning systems—would enable users greater freedom to embed the Wekinator within their own systems.

Throughout these changes, we plan to maintain the general-purpose nature of the Wekinator so that it remains possible to apply the system to non-musical problems. We would like to work with non-musical users, for example users designing interactive art or gaming systems, to better understand their needs and ensure that the Wekinator supports this work well. We would also like to apply the Wekinator to real-time audio analysis problems in computer music performance, an initial goal of our work that has not yet been realized.

An additional set of improvements will focus on continuing to improve the usefulness of the Wekinator as a tool for conducting research in interactive supervised learning in real-time domains. We will strive to make it as easy as possible for ourselves and other researchers to add new learning algorithms, new modes of feedback to users, and new mechanisms for logging and understanding users' actions with the system.

10.2.2 Research Directions in HCI, Machine Learning, and Computer Music

This work has helped to formulate several research directions surrounding the improvement of software systems for end-user interactive machine learning and the application of interactive machine learning to new problems and domains. First, our work has shown that users applying interactive machine learning may or may not be concerned with the generalization accuracy of models, and even when generalization accuracy is important, it may be only one of several characteristics that users care about. Through future work investigating what characteristics of trained models may be important to users across different application domains, we might arrive at a better understanding of how to design algorithms that create models with these characteristics and how to design interfaces that allow users to evaluate models against these characteristics. Furthermore, future work should investigate whether learning algorithms capable of prioritizing training accuracy at the expense of generalization accuracy may be better-suited to interactive applications in which users exercise control over the learning problem primarily through modifications to the training set.

Other future work might combine HCI and machine learning approaches in other ways. For example, research into the design of algorithms with parameters whose effects are clearly understandable to users and likely to affect dimensions of a model that users care about could result in more usable algorithms. Alternatively, work that combines theoretical understanding of algorithms and novel user interface techniques might better enable users to employ the parameters of standard algorithms to more effectively improve models against their criteria for success.

Our findings suggest that the degree to which users' goals for supervised learning are flexible impacts the ways in which they may use interaction to build models to achieve their goals. Future work might study users completing different types of tasks in a more controlled environment to better understand how users' interface and interaction requirements differ according to the degree of this flexibility. Additionally, future work might consider other problem domains in which users' goals are relatively fixed—including domains where conventional machine learning is often applied—and investigate the extent to which interactive approaches can still exploit the aspects of the learning problem (e.g., number and nature of classes, sources of features) that remain flexible.

Another research direction concerns the creation and analysis of additional interface mechanisms to provide richer and more useful feedback to users, including users who are machine learning novices. In our observations, users employed interactive evaluation of models for a variety of purposes, including to assess the quality of trained models against their evaluation criteria, to determine which actions they might take to improve those models, to learn what might or might not be possible to accomplish using the given algorithms, and to learn which of their actions were most effective in creating the types of models they wanted. These types of feedback were all important to users, and future work might investigate new user interfaces to provide this feedback more efficiently and effectively, to provide complementary

information, or to provide information tailored to novice users who are just learning how to use interactive machine learning effectively.

Our research has highlighted the ability for interactive machine learning to effectively support creativity and an embodied approach to building interactive systems. We are excited about the prospects to explore how users in other creative and interactive problem domains, including interactive art and game design, might benefit from interactive supervised learning tools. Additionally, we observed the capability for interactive supervised learning to cause users to reflect on their own abilities and actions; for example, users sometimes discovered through their failure to build an accurate classifier that the training data they created was not consistent or clear. Future work could therefore investigate the potential of interactive supervised learning in pedagogical or therapy applications where users may benefit from mechanisms for learning about the consistency of their actions.

Finally, further work with composers, instrument designers, musicians, and music students will likely illuminate new avenues for interactive supervised learning to be applied more effectively and more broadly to creative work in computer music. New algorithms could be designed to incorporate behaviors and tunable parameters tailored specifically for application to the design of new musical instruments. New interfaces could also be built to allow new types of supervised learning systems. For example, systems with hierarchical relationships between models, in which the outputs of some models serve as input features to other models, could be useful for problems in which more sophisticated segmentation mechanisms are necessary, as well as for allowing synthesis or other music-producing processes to be controlled in new ways. Or, interfaces and algorithms could be constructed that enable the trained models to smoothly evolve or change over time in response to human actions or random processes. In any case, we hope to continue to engage composers and musicians in participatory design of these systems, as we have found our work with users to be crucial in improving and evaluating our software, understanding their priorities for incorporating technology into their work, and identifying key research questions pertaining to interactive supervised learning in music and other domains.

10.3 Conclusion

Our work has emphasized the importance of considering the human context of machine learning practice. By providing users with the ability to engage in a wider variety of interactions with supervised learning algorithms, the software we have created has enabled people to put standard learning algorithms to use to do their work more efficiently and effectively, and to imagine and accomplish goals that were not possible with the tools previously available to them. Our work has shown that these interactions can empower end users to build interactive systems for themselves and for others while employing an approach to design that leverages their domain knowledge, embodied expertise, and understanding of the context in which the trained models will be used. Furthermore, these interactions can inspire users to consider new design

possibilities and engage in self-reflection and growth as machine learning users and creative beings.

In our work, the principles and methodologies of HCI research, the computational possibilities presented by machine learning, and the knowledge, values, and ideas of domain experts have been instrumental in shaping our research questions about how the computational and interactive affordances of learning algorithms can be exploited to meet real-world application requirements and support users' values. Having demonstrated the feasibility and usefulness of applying interactive supervised learning to the musical problems studied in this thesis, we are excited by the opportunities for future work that, by continuing to build on the intersection of HCI, machine learning, and application domain knowledge, will lead to new algorithms, interfaces, and interactions capable of supporting even more users in applying supervised learning more effectively and to more problems.

Bibliography

Allen, J. M., Asselin, P. K., and Foulds, R. 2003. American sign language finger spelling recognition system. In *Proceedings of the IEEE 29th Bioengineering Conference*. 285–286.

Amershi, S., Fogarty, J., Kapoor, A., and Tan, D. 2009. Overview-based example selection in end-user interactive concept learning. In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST '09)*. 247–256.

Amershi, S., Fogarty, J., Kapoor, A., and Tan, D. 2010. Examining multiple potential models in end-user interactive concept learning. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 1357–1360.

Andrews, R. 2007. Reactable tactile synth catches Björk's eye–and ear. *Wired Magazine*, 8 September 2007. `http://www.wired.com/entertainment/music/news/2007/08/bjork_reacTable`. Also available at `http://www.cs.princeton.edu/~fiebrink/thesis/resources.html`.

Apple Inc. 2008. Apple portables: About the sudden motion sensor. `http://support.apple.com/kb/HT1935`. Also available at `http://www.cs.princeton.edu/~fiebrink/thesis/resources.html`.

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., and Zaharia, M. 2010. A view of cloud computing. *Communications of the ACM 53*, 50–58.

Armstrong, N. 2006. An enactive approach to digital musical instrument design. Ph.D. thesis, Princeton University, Princeton, NJ, USA.

Baker, K., Bhandari, A., and Thotakura, R. 2009. An interactive automatic document classification prototype. In *Proceedings of the Third Workshop on Human-Computer Interaction and Information Retrieval (HCIR 2009)*. 30–33.

Balcan, M.-F., Hanneke, S., and Vaughan, J. 2010. The true sample complexity of active learning. *Machine Learning 80*, 111–139.

Beale, M. H., Hagan, M. T., and Demuth, H. B. 2010. *Neural Network Toolbox 7*. The Mathworks, Inc., Natick, MA, USA.

Bencina, R. 2005. The Metasurface: Applying natural neighbor interpolation to two-to-many mapping. In *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*. Vancouver, BC, Canada, 101–104.

Bergstra, J., Casagrande, N., Erhan, D., Eck, D., and Kégl, B. 2006. Aggregate features and AdaBoost for music classification. *Machine Learning 65,* 2, 473–484.

Bevilacqua, F., Müller, R., and Schnell, N. 2005. MnM: A Max/MSP mapping toolbox. In *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*. 85–88.

Billard, A., Calinon, S., Dillman, R., and Schaal, S. 2008. Robot programming by demonstration. In *Handbook of Robotics*, B. Siciliano and O. Khatib, Eds. Springer, 1371–1394.

Bischoff, J., Gold, R., and Horton, J. 1978. Music for an interactive network of microcomputers. *Computer Music Journal 2,* 3, 24–29.

Bishop, C. M. 2007. *Pattern Recognition and Machine Learning*, 2nd ed. Springer.

Blumer, A., Ehrenfeucht, A., Haussler, D., and Warmuth, M. 1986. Classifying learnable geometric concepts with the Vapnik-Chervonenkis dimension. In *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing (STOC '86)*. 273–282.

Boie, B., Mathews, M., and Schloss, A. 1989. The Radio Drum as a synthesizer controller. In *Proceedings of the International Computer Music Conference (ICMC)*. 42–45.

Bongers, B. 2000. Physical interfaces in the electronic arts. In *Trends in Gestural Control of Music*, M. M. Wanderley and M. Battier, Eds. IRCAM—Centre Pompidou, 41–70.

Boulanger, R., Ed. 2000. *The Csound Book: Perspectives in Software Synthesis, Sound Design, Signal Processing, and Programming.* The MIT Press, Cambridge, Massachusetts.

Brecht, B. and Garnett, G. E. 1995. Conductor following. In *Proceedings of the International Computer Music Conference (ICMC)*. 185–186.

Brent, W. 2010. A timbre analysis and classification toolkit for Pure Data. In *Proceedings of the International Computer Music Conference (ICMC)*.

Brown, C. and Bischoff, J. 2002. Indigenous to the net: Early network music bands in the San Francisco Bay area. *Crossfade*. `http://crossfade.walkerart.org/brownbischoff/IndigenoustotheNetPrint.html`. Also available at `http://www.cs.princeton.edu/~fiebrink/thesis/resources.html`.

Burges, C. 1998. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery 2,* 2, 121–167.

Cano, P., Koppenberger, M., and Wack, N. 2005. Content-based music audio recommendation. In *Proceedings of the 13th Annual ACM International Conference on Multimedia (MULTIMEDIA '05)*. 211–212.

Chadabe, J. 1997. *Electric Sound: The Past and Promise of Electronic Music*. Prentice Hall, Upper Saddle River, New Jersey.

Chadabe, J. 2002. The limitations of mapping as a structural descriptive in electronic instruments. In *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*.

Chang, C.-C. and Lin, C.-J. 2001. *LIBSVM: A library for support vector machines*. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

Chowning, J. 1973. The synthesis of complex audio spectra by means of frequency modulation. *Journal of the Audio Engineering Society 21,* 7.

Cont, A., Coduys, T., and Henry, C. 2004. Real-time gesture mapping in Pd environment using neural networks. In *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*. 39–42.

Cook, P. R. 2001. Principles for designing computer music controllers. In *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*.

Cook, P. R. 2005. Real-time performance controllers for synthesized singing. In *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*. Vancouver, BC, Canada, 236–237.

Cook, P. R. and Scavone, G. P. 1999. The Synthesis ToolKit (STK). In *Proceedings of the International Computer Music Conference (ICMC)*. 164–166.

Cook, P. R. and Trueman, D. 1998. NBody: Interactive multidirectional musical instrument body radiation simulations, and a database of measured impulse responses. In *Proceedings of the International Computer Music Conference (ICMC)*.

Cypher, A. 1993. *Watch What I Do: Programming by Demonstration*. MIT Press, Cambridge, Massachusetts.

Dannenberg, R. 1989. Real-time scheduling and computer accompaniment. In *Current Research in Computer Music*, M. Mathews and J. Pierce, Eds. The MIT Press, Cambridge, Massachusetts, 225–261.

Dannenberg, R., Birmingham, W., Tzanetakis, G., Meek, C., Hu, N., and Pardo, B. 2004. The MUSART testbed for query-by-humming evaluation. *Computer Music Journal 28,* 2, 34–48.

Dobrian, C. and Bevilacqua, F. 2003. Gestural control of music: Using the vicon 8 motion capture system. In *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*. 161–163.

Doornbusch, P. 2004. Computer sound synthesis in 1951: The music of CSIRAC. *Computer Music Journal 28,* 1, 10–25.

Drummond, J. 2009. Understanding interactive systems. *Organised Sound 14,* 2, 124–133.

Dunne, J. 2007. Monome 40h multi-purpose hardware controller (product review). *Computer Music Journal 31,* 3, 92–94.

Eck, D., Bertin-Mahieux, T., and Lamere, P. 2007. Autotagging music using supervised machine learning. In *Proceedings of the International Conference on Music Information Retrieval (ISMIR)*. Vienna, Austria, 367–368.

Eerola, T. and Toiviainen, P. 2004. MIR in Matlab: The MIDI toolbox. In *Proceedings of the International Conference on Music Information Retrieval (ISMIR)*. 22–27.

Eisenberg, B. B. 1992. On the sample complexity of PAC-learning using random and chosen examples. M.S. thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts.

Ellis, D. P. and Poliner, G. E. 2006. Classification-based melody transcription. *Machine Learning 65,* 2-3, 439–456.

Fails, J. A. and Olsen, Jr., D. R. 2003. Interactive machine learning. In *Proceedings of the International Conference on Intelligent User Interfaces (IUI '03)*. 39–45.

Fels, S. S. and Hinton, G. E. 1995. Glove-TalkII: An adaptive gesture-to-formant interface. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. Denver, CO, 456–463.

Fiebrink, R. 2006. An exploration of feature selection as a tool for optimizing musical genre classification. M.A. thesis, McGill University, Montréal, QC, Canada.

Fiebrink, R. 2010. Real-time interaction with supervised learning. In *Extended Abstracts of the SIGCHI Conference on Human Factors in Computing Systems*. 2935–2938.

Fiebrink, R., Cook, P. R., and Trueman, D. 2009. Play-along mapping of musical controllers. In *Proceedings of the International Computer Music Conference (ICMC)*. Montréal, QC, Canada.

Fiebrink, R. and Fujinaga, I. 2006. Feature selection pitfalls and music classification. In *Proceedings of the International Conference on Music Information Retrieval (ISMIR)*. Victoria, BC, Canada.

Fiebrink, R., Schedel, M., and Threw, B. 2010. Constructing a personalizable gesture-recognizer infrastructure for the K-Bow. In *Proceedings of the 3rd International Conference on Music and Gesture (MG3)*.

Fiebrink, R., Trueman, D., Britt, C., Nagai, M., Kaczmarek, K., Early, M., Daniel, M. R., Hege, A., and Cook, P. R. 2010. Toward understanding human-computer interaction in composing the instrument. In *Proceedings of the International Computer Music Conference (ICMC)*.

Fiebrink, R., Trueman, D., and Cook, P. R. 2009. A meta-instrument for interactive, on-the-fly machine learning. In *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*.

Fiebrink, R., Wang, G., and Cook, P. R. 2007. Don't forget the laptop: Using native input capabilities for expressive musical control. In *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*.

Fiebrink, R., Wang, G., and Cook, P. R. 2008. Foundations for on-the-fly learning in the ChucK programming language. In *Proceedings of the International Computer Music Conference (ICMC)*.

Fildes, J. 2008. 'Oldest' computer music unveiled. *BBC News*. `http://news.bbc.co.uk/2/hi/technology/7458479.stm`.

Flesch, C. 2000. *The Art of Violin Playing: Book One*. Carl Fischer, New York, NY, USA.

Fogarty, J., Tan, D., Kapoor, A., and Winder, S. 2008. CueFlik: Interactive concept learning in image search. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 29–38.

Freund, Y. and Schapire, R. E. 1997. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences 55,* 1, 119–139.

Garner, W. R. 1974. *The Processing of Information and Structure*. Psychology Press.

Ghias, A., Logan, J., Chamberlin, D., and Smith, B. 1995. Query by humming: Musical information retrieval in an audio database. In *Proceedings of the Third ACM International Conference on Multimedia*. 231–236.

Gibbs, R. W. 2006. *Embodiment and Cognitive Science*. Cambridge University Press, New York, NY, USA.

Gibson, J. J. 1979. *The Ecological Approach to Visual Perception*. Houghton Mifflin, Boston, MA, USA.

Guyon, I. and Elisseeff, A. 2003. An introduction to variable and feature selection. *Journal of Machine Learning Research 3*, 1157–1182.

Hahn, T. and Bahn, C. 2003. Pikapika—The collaborative composition of an interactive sonic character. *Organised Sound 7,* 3, 229–238.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. 2009. The WEKA data mining software: An update. *SIGKDD Explorations 11,* 1, 10–18.

Harrison, C. and Hudson, S. E. 2008. Scratch input: Creating large, inexpensive, unpowered and mobile finger input surfaces. In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST '08)*. 205–208.

Hartmann, B., Abdulla, L., Mittal, M., and Klemmer, S. R. 2007. Authoring sensor-based interactions by demonstration with direct manipulation and pattern recognition. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 145–154.

Hunt, A. and Kirk, R. 2000. Mapping strategies for musical performance. In *Trends in Gestural Control of Music*, M. M. Wanderley and M. Battier, Eds. IRCAM—Centre Pompidou.

Hunt, A. and Wanderley, M. M. 2002. Mapping performer parameters to synthesis engines. *Organised Sound 7,* 2, 97–108.

Hunt, A., Wanderley, M. M., and Paradis, M. 2002. The importance of parameter mapping in electronic instrument design. In *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*.

ISO 9241-11:1998. 1998. *Ergonomics of Human System Interaction: Guidance on Usability*. ISO, Geneva, Switzerland.

Jacob, R. J. K., Sibert, L. E., McFarlane, D. C., and Mullen, Jr., M. P. 1994. Integrality and separability of input devices. *ACM Transactions on Computer-Human Interaction (TOCHI) 1*, 3–26.

JazzMutant. 2009. *Lemur v2.0 User Guide*. JazzMutant.

Jehan, T. and Schoner, B. 2001. An audio-driven perceptually meaningful timbre synthesizer. In *Proceedings of the International Computer Music Conference (ICMC)*. Havana, Cuba.

Jelinek, F., Bahl, L. R., and Mercer, R. L. 1975. Design of a linguistic statistical decoder for the recognition of continuous speech. *IEEE Transactions on Information Theory 21,* 3, 250–256.

Jordà, S., Geiger, G., Alonso, M., and Kaltenbrunner, M. 2007. The reacTable: Exploring the synergy between live music performance and tabletop tangible interfaces. In *Proceedings of the 1st International Conference on Tangible and Embedded Interaction (TEI '07)*. 139–146.

Kapoor, A., Lee, B., Tan, D., and Horvitz, E. 2010. Interactive optimization for steering machine classification. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 1343–1352.

Kapur, A. 2007. Digitizing North Indian performance: Extension and preservation using multimodal sensor systems, machine learning, and robotics. Ph.D. thesis, University of Victoria, Victoria, BC, Canada.

Keith McMillen Instruments 2009a. *Attaching the K-Bow Emitter*. Keith McMillen Instruments.

Keith McMillen Instruments 2009b. *K-Bow Reference Manual*. Keith McMillen Instruments.

Klemmer, S., Hartmann, B., and Takayama, L. 2006. How bodies matter: Five themes for interaction design. In *Proceedings of the 6th Conference on Designing Interactive Systems*. 140–149.

Knapp, R. B. and Tanaka, A. 2002. Multimodal interaction in music using the electromyogram and relative position sensing. In *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*. Dublin, Ireland.

Lartillot, O. and Toiviainen, P. 2007. MIR in Matlab (II): A toolbox for musical feature extraction from audio. In *Proceedings of the International Conference on Music Information Retrieval (ISMIR)*. 237–244.

Lau, T., Wolfman, S. A., Domingos, P., and Weld, D. S. 2001. Learning repetitive text-editing proceedures with SMARTedit. In *Your Wish is My Command: Programming By Example*, H. Lieberman, Ed. Morgan Kaufmann, Chapter 11, 209–226.

Lee, M., Freed, A., and Wessel, D. 1991. Real-time neural network processing of gestural and acoustic signals. In *Proceedings of the International Computer Music Conference (ICMC)*. 277–280.

Lee, M. A., Garnett, G., and Wessel, D. 1992. An adaptive conductor follower. In *Proceedings of the International Computer Music Conference (ICMC)*. 454–455.

Lew, M., Sebe, N., Djeraba, C., and Jain, R. 2006. Content-based multimedia information retrieval: State of the art and challenges. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMCCAP) 2,* 1, 1–19.

Lewis, D. and Gale, W. 1994. A sequential algorithm for training text classifiers. In *Proceedings of the Seventeenth Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval*. 3–12.

Lewis, G. 2000. Too many notes: Computers, complexity and culture in *Voyager*. *Leonardo Music Journal 10*, 33–39.

Lieberman, H. 2001. Introduction. In *Your Wish is My Command: Programming By Example*, H. Lieberman, Ed. Morgan Kaufmann, 1–6.

Lindemann, E., Starkier, M., and Dechelle, F. 1990. The IRCAM musical workstation: Hardware overview and signal processing features. In *Proceedings of the International Computer Music Conference (ICMC)*. San Francisco, CA, 132–135.

Linden, G., Smith, B., and York, J. 2003. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing 7,* 1, 76 – 80.

Lippe, C. 1997. Music for piano and computer: A description. *Information Processing Soceity of Japan, SIG Notes 97,* 122, 33–38.

Lippe, C. 2002. Real-time interaction among composers, performers, and computer systems. *Information Processing Soceity of Japan, SIG Notes 123*, 1–6.

Logan, B. 2000. Mel frequency cepstral coefficients for music modeling. In *Proceedings of the International Symposium on Music Information Retrieval (ISMIR)*. Vol. 28.

McCartney, J. 2002. Rethinking the computer music language: SuperCollider. *Computer Music Journal 26,* 4, 61–68.

McGrenere, J. and Ho, W. 2000. Affordances: Clarifying and evolving a concept. In *Proceedings of Graphic Interface (GI '00)*. Montréal, QC, Canada.

McKay, C. and Fujinaga, I. 2009. jMIR: Tools for automatic music classification. In *Proceedings of the International Computer Music Conference (ICMC)*. 65–68.

McMillen, K. A. 2008. Stage-worthy sensor bows for stringed instruments. In *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*.

Merrill, D. J. and Paradiso, J. A. 2005. Personalization, expressivity, and learnability of an implicit mapping strategy for physical interfaces. In *Extended Abstracts of the SIGCHI Conference on Human Factors in Computing Systems*.

Microsoft Corporation. 2010. Set up speech recognition. `http://windows.microsoft.com/en-US/windows7/Set-up-Speech-Recognition`.

Miranda, E. and Wanderley, M. M. 2006. *New Digital Musical Instruments: Control and Interaction Beyond the Keyboard*. A-R Editions, Inc., Middleton, Wisconsin.

Modler, P. 2000. Neural networks for mapping gestures to sound synthesis. In *Trends in Gestural Control of Music*, M. M. Wanderley and M. Battier, Eds. IRCAM—Centre Pompidou.

Modler, P., Hofmann, F., and Zannos, I. 1998. Gesture recognition by neural networks and the expression of emotions. *IEEE Transactions on Systems, Man, and Cybernetics 2*, 1072–1075.

Moon, B. 1997. Score following and real-time signal processing strategies in open-form compositions. *Information Processing Soceity of Japan, SIG Notes 97,* 122, 12–19.

Morris, D., Simon, I., and Basu, S. 2008. Exposing parameters of a trained dynamic model for interactive music creation. In *AAAI'08: Proceedings of the 23rd National Conference on Artificial Intelligence*. 784–791.

Münch, S., Kreuziger, J., Kaiser, M., and Dillmann, R. 1994. Robot programming by demonstration (RPD): Using machine learning and user interaction methods for the development of easy and comfortable robot programming systems. In *Proceedings of the 24th International Symposium on Industrial Robots (ISIR '94)*.

Murata, K., Nakadai, K., Yohii, K., Takeda, R., Torii, T., Okuno, H. G., Hasegawa, Y., and Tsujino, H. 2008. A robot singer with music recognition based on real-time beat tracking. In *Proceedings of the International Conference on Music Information Retrieval (ISMIR)*. Philadelphia, PA, USA, 199–204.

Nagai, M. 2010a. Bio/CV. `http://michellenagai.com/Site/Bio_CV.html`. Also available at `http://www.cs.princeton.edu/~fiebrink/thesis/resources.html`.

Nagai, M. 2010b. MARtLET. `http://michellenagai.com/Site/MARtLET.html`. Also available at Also available at `http://www.cs.princeton.edu/~fiebrink/thesis/resources.html`.

Nakra, T. M. 2000. Inside the conductor's jacket: Analysis, interpretation, and musical synthesis of expressive gesture. Ph.D. thesis, Massachusetts Institute of Technology Media Laboratory, Cambridge, Massachusetts.

Nichols, E., Morris, D., and Basu, S. 2009. Data-driven exploration of musical chord sequences. In *Proceedings of the International Conference on Intelligent User Interfaces (IUI '09)*. 227–236.

Norman, D. A. 1988. *The Psychology of Everyday Things*. Basic Books, New York, NY, USA.

Norman, D. A. and Draper, S. W. 1986. *User-Centered System Design: New Perspectives on Human-Computer Interaction*. Lawrence Erlbaum Associates, Inc., Hillsdale, New Jersey.

Nuance Communications, Inc. 2010. *Dragon NaturallySpeaking 10: End-User Workbook*.

Opolko, F. and Wapnick, J. 1989. *Mcgill University master samples*. McGill University, Faculty of Music.

Paine, G., Ed. 2009. *Organised Sound 14, 2*.

Paradiso, J. A. and Gershenfeld, N. 1997. Musical applications of electric field sensing. *Computer Music Journal 21,* 2, 69–89.

Pareles, J. 2007. Pay what you want for this article. *The New York Times*, 9 December.

Patel, K., Bancroft, N., Drucker, S. M., Fogarty, J., Ko, A. J., and Landay, J. 2010. Gestalt: Integrated support for implementation and analysis in machine learning. In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST '10)*. 37–46.

Peiper, C., Warden, D., and Garnett, G. 2003. An interface for real-time classification of articulations produced by violin bowing. In *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*.

Percival, G. and Tzanetakis, G. 2006. *Marsyas User Manual for Version 0.2*.

Platt, J. 1999. Fast training of support vector machines using sequential minimal optimization. In *Advances in Kernel Methods: Support Vector Learning*, B. Schoelkopf, C. Burges, and A. Smola, Eds. MIT Press, Cambridge, Massachusetts, 185–208.

Poliner, G. and Ellis, D. 2005. A classification approach to melody transcription. In *Proceedings of the International Conference on Music Information Retrieval (ISMIR)*. 161–166.

Puckette, M. 1991. Combining event and signal processing in the MAX graphical programming environment. *Computer Music Journal 15,* 3, 68–77.

Quinlan, J. R. 1993. *C4.5: Programs for Machine Learning*. Morgan Kaufmann.

Raphael, C. 2010. Music Plus One and machine learning. In *Proceedings of the Twenty-Seventh International Conference on Machine Learning (ICML)*.

Rapid-I. 2010. *RapidMiner 5.0 Manual*. Rapid-I GmbH.

Rasamimanana, N., Fléty, E., and Bevilacqua, F. 2005. Gesture analysis of violin bow strokes. In *Proceedings of Gesture Workshop 2005 (GW05)*. 145–155.

Reas, C. and Fry, B. 2007. *Processing: A Programming Handbook for Visual Designers and Artists*. The MIT Press, Cambridge, Massachusetts.

Reid, G. 2001. Sounds of the '80s part 2: The Yamaha DX1 and its successors. *Sound On Sound*. `http://www.soundonsound.com/sos/sep01/articles/retrofmpt2.asp`. Also available at `http://www.cs.princeton.edu/~fiebrink/thesis/resources.html`.

Resnick, M., Myers, B., Nakakoji, K., Shneiderman, B., Pausch, R., Selker, T., and Eisenberg, M. 2005. Design principles for tools to support creative thinking. In *Report of Workshop on Creativity Support Tools*. Washington, DC, USA.

Ross, M. E., Zhou, X., Song, G., Shurtleff, S. A., Girtman, K., Williams, W. K., Liu, H.-C., Mahfouz, R., Raimondi, S. C., Lenny, N., Patel, A., and Downing, J. R. 2003. Classification of pediatric acute lymphoblastic leukemia by gene expression profiling. *Blood 102,* 8 (October), 2951–2959.

Rowe, R. 1993. *Interactive Music Systems*. The MIT Press, Cambridge, Massachusetts.

Rowe, R., Garton, B., Desain, P., Honing, H., Dannenberg, R., Jacobs, D., Pope, S. T., Puckette, M., Lippe, C., Settel, Z., and Lewis, G. 1993. Editor's notes: Putting Max in perspective. *Computer Music Journal 17,* 2, 3–11.

Russell, S. and Norvig, P. 2003. *Artificial Intelligence: A Modern Approach*, 2nd ed. Prentice Hall.

Schapire, R. E. 2003. The boosting approach to machine learning: An overview. *Lecture Notes in Statistics*, 149–172.

Schnell, N. and Battier, M. 2002. Introducing composed instruments, technical and musicological implications. In *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*.

Shilman, M., Tan, D., and Simard, P. 2006. CueTIP: A mixed-initiative interface for correcting handwriting errors. In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST '06)*. 323–332.

Shneiderman, B. 2000. Creating creativity: user interfaces for supporting innovation. *ACM Transactions on Computer-Human Interaction (TOCHI) 7*, 114–138.

Shneiderman, B. 2002. *Leonardo's Laptop: Human Needs and the New Computing Technologies*. The MIT Press, Cambridge, Massachusetts.

Shneiderman, B. 2007. Creativity support tools: accelerating discovery and innovation. *Communications of the ACM 50*, 20–32.

Sinyor, E., McKay, C., Fiebrink, R., McEnnis, D., and Fujinaga, I. 2005. Beatbox classification using ACE. In *Proceedings of the International Computer Music Conference (ICMC)*. London, UK.

Skriver Hansen, A., Overholt, D., Burleson, W., and Jensen, C. M. 2009. Pendaphonics: A tangible pendulum-based sonic interaction experience. In *Proceedings of the 3rd International Conference on Tangible and Embedded Interaction (TEI '09)*. 153–160.

Slonimsky, N. 1998. *Webster's New World Dictionary of Music*. Wiley Publishing, Inc.

Smith, J. O. 1986. Efficient simulation of the reed-bore and bow-string mechanisms. In *Proceedings of the International Computer Music Conference (ICMC)*.

Snyder, J. 2010. Exploration of an adaptable just intonation system. D.M.A. thesis, Columbia University, New York City.

Steiner, H.-C. 2005. [hid] toolkit: A unified framework for instrument design. In *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*. 140–143.

Stiefel, V., Trueman, D., and Cook, P. R. 2004. Re-coupling: The uBlotar synthesis instrument and the sHowl speaker-feedback controller. In *Proceedings of the International Computer Music Conference (ICMC)*.

Talbot, J., Lee, B., Kapoor, A., and Tan, D. S. 2009. EnsembleMatrix: Interactive visualization to support machine learning with multiple classifiers. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 1283–1292.

Tanaka, A. 2000. Musical performance practice on sensor-based instruments. In *Trends in Gestural Control of Music*, M. M. Wanderley and M. Battier, Eds. IRCAM—Centre Pompidou, 389–405.

Teixeira, L. F., Martins, L. G., Lagrange, M., and Tzanetakis, G. 2008. MarsyasX: Multimedia dataflow processing with implicit patching. In *Proceedings of the 16th ACM International Conference on Multimedia*. 873–876.

Tindale, A. 2004. Classifying snare drum sounds using neural networks. M.A. thesis, McGill University, Montréal, QC, Canada.

Trueman, D. 2007. Why a laptop orchestra? *Organised Sound 12,* 2, 171–179.

Trueman, D. 2010a. Clapping Machine Music Variations: A composition for acoustic/laptop ensemble. In *Proceedings of the International Computer Music Conference (ICMC)*. New York, NY, USA.

Trueman, D. 2010b. Clapping Machine Music Variations program notes. `http://www.music.princeton.edu/~nbritt/PLOrk-spring2010/`. Also available at `http://www.cs.princeton.edu/~fiebrink/thesis/resources.html`.

Trueman, D., Bahn, C., and Cook, P. R. 2000. Alternative voices for electronic sound. In *Proceedings of the International Computer Music Conference (ICMC)*.

Trueman, D. and Cook, P. R. 2000. BoSSA: The deconstructed violin reconstructed. *Journal of New Music Research 29,* 2, 121–130.

Trueman, D., Cook, P. R., Smallwood, S., and Wang, G. 2006. PLOrk: The Princeton laptop orchestra, year 1. In *Proceedings of the International Computer Music Conference (ICMC)*.

Tsandilas, T., Letondal, C., and Mackay, W. E. 2009. Musink: Composing music through augmented drawing. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 819–828.

Turnbull, D., Barrington, L., Torres, D., and Lanckriet, G. R. G. 2008. Semantic annotation and retrieval of music and sound effects. *IEEE Transactions on Audio, Speech, and Language Processing 16,* 2, 467–476.

Tzanetakis, G. and Cook, P. R. 2000. MARSYAS: A framework for audio analysis. *Organised Sound 4,* 3, 169–175.

Tzanetakis, G., Essl, G., and Cook, P. R. 2001a. Audio analysis using the discrete wavelet transform. In *Proceedings of the Conference in Acoustics and Music Theory Applications*.

Tzanetakis, G., Essl, G., and Cook, P. R. 2001b. Automatic musical genre classification of audio signals. In *Proceedings of the International Symposium on Music Information Retrieval (ISMIR)*. Bloomington, Indiana, USA, 205–210.

Tzanetakis, G. and Lemstrom, K. 2007. Marsyas-0.2: A case study in implementing music information retrieval systems. *Intelligent Music Information Systems. IGI Global*.

USB Implementers' Forum. 2001. *Device Class Definition for Human Interface Devices (HID): Version 1.11*.

Valiant, L. G. 1984. A theory of the learnable. *Communications of the ACM 27,* 11, 1134–1142.

Van Nort, D. 2009. Instrumental listening: Sonic gesture as design principle. *Organised Sound 14,* 2, 177–187.

Vapnik, V. N. 1999. An overview of statistical learning theory. *IEEE Transactions on Neural Networks 10,* 5, 988–999.

Vertanen, K. 2009. Efficient correction interfaces for speech recognition. Ph.D. thesis, University of Cambridge.

Vertegaal, R. and Eaglestone, B. 1996. Comparison of input devices in an ISEE direct timbre manipulation task. *Interacting with Computers 8,* 1, 13–30.

Vredenburg, K. 1999. Increasing ease of use. *Communications of the ACM 42,* 5, 67–71.

Vredenburg, K., Mao, J.-Y., Smith, P. W., and Carey, T. 2002. A survey of user-centered design practice. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 471–478.

Waisvisz, M. 1985. The Hands: A set of remote midi-controllers. In *Proceedings of the International Computer Music Conference (ICMC)*. 313–318.

Wanderley, M. M. 2002. Quantitative analysis of non-obvious performer gestures. In *Gesture and Sign Language in Human-Computer Interaction*, I. Wachsmuth and T. Sowa, Eds. Lecture Notes in Computer Science, vol. 2298. Springer Berlin / Heidelberg, 241–253.

Wanderley, M. M. and Orio, N. 2002. Evaluation of input devices for musical expression: Borrowing tools from HCI. *Computer Music Journal 26,* 3, 62–76.

Wang, G. and Cook, P. R. 2003. ChucK: A concurrent, on-the-fly audio programming language. In *Proceedings of the International Computer Music Conference (ICMC)*.

Wang, G., Essl, G., and Penttinen, H. 2008. Do mobile phones dream of electric orchestras? In *Proceedings of the International Computer Music Conference (ICMC)*.

Wang, G., Fiebrink, R., and Cook, P. R. 2007. Combining analysis and synthesis in the ChucK programming language. In *Proceedings of the International Computer Music Conference (ICMC)*.

Weitekamp, R. 2009. SmackTop 0.1. `http://altitudesickness.noisepages.com/2009/10/smacktop0_/`.

Wessel, D. 1991. Instruments that learn, refined controllers, and source model loudspeakers. *Computer Music Journal 15,* 4, 81–86.

Wessel, D. 2006. An enactive approach to computer music performance. In *Le Feedback dans la Creation Musical*, Y. Orlarey, Ed. Studio Gramme, Lyon, France, 93–98.

West, K. and Cox, S. 2005. Finding an optimal segmentation for audio genre classification. In *Proceedings of the International Conference on Music Information Retrieval (ISMIR)*.

Wilcox, R. R. 2010. *Fundamentals of Modern Statistical Methods: Substantially Improving Power and Accuracy*. Springer.

Wilson, A. and Bobick, A. 2000. Realtime online adaptive gesture recognition. In *Proceedings of the 15th International Conference on Pattern Recognition*. 270–275.

Winkler, T. 2002. Fusing movement, sound, and video in *Falling Up*, an interactive dance/theatre production. In *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*. Dublin, Ireland.

Witten, I. H. and Frank, E. 2005. *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd ed. Morgan Kaufmann, New York, NY, USA.

Wright, M. and Freed, A. 1997. Open Sound Control: A new protocol for communicating with sound synthesizers. In *Proceedings of the International Computer Music Conference (ICMC)*.

Yoshii, K., Goto, M., Komatani, K., Ogata, T., and Okuno, H. 2006. Hybrid collaborative and content-based music recommendation using probabilistic model with latent user preferences. In *Proceedings of the International Conference on Music Information Retrieval (ISMIR)*. Victoria, BC, Canada.

Young, D. 2008. Classification of common violin bowing techniques using gesture data from a playable measurement system. In *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*.