COS 511: Foundations of Machine Learning

Rob Schapire
Scribe: Rob Schapire

Lecture #1
February 4, 2003

1 What is Machine Learning?

Machine learning studies computer algorithms for learning to do stuff. We might, for instance, be interested in learning to complete a task, or to make accurate predictions, or to behave intelligently. The learning that is being done is always based on some sort of observations or data, such as examples (the most common case in this course), direct experience, or instruction. So in general, machine learning is about learning to do better in the future based on what was experienced in the past.

The emphasis of machine learning is on *automatic* methods. In other words, the goal is to devise learning algorithms that do the learning automatically without human intervention or assistance. The machine learning paradigm can be viewed as "programming by example." Often we have a specific task in mind, such as spam filtering. But rather than program the computer to solve the task directly, in machine learning, we seek methods by which the computer will come up with its own program based on examples that we provide.

Machine learning is a core subarea of artificial intelligence. It is very unlikely that we will be able to build any kind of intelligent system capable of any of the facilities that we associate with intelligence, such as language or vision, without using learning to get there. These tasks are otherwise simply too difficult to solve. Further, we would not consider a system to be truly intelligent if it were incapable of learning since learning is at the core of intelligence.

Although a subarea of AI, machine learning also intersects broadly with other fields, especially statistics, but also mathematics, physics, theoretical computer science and more.

2 Examples of Machine Learning Problems

There are many examples of machine learning problems. Much of this course will focus on classification problems in which the goal is to categorize objects into a fixed set of categories. Here are several examples:

- optical character recognition: categorize images of handwritten characters by the letters represented
- face detection: find faces in images (or indicate if a face is present)
- spam filtering: identify email messages as spam or non-spam
- topic spotting: categorize news articles (say) as to whether they are about politics, sports, entertainment, etc.
- spoken language understanding: within the context of a limited domain, determine the meaning of something uttered by a speaker to the extent that it can be classified into one of a fixed set of categories

- medical diagnosis: diagnose a patient as a sufferer or non-sufferer of some disease
- customer segmentation: predict, for instance, which customers will respond to a particular promotion
- fraud detection: identify credit card transactions (for instance) which may be fraudulent in nature
- weather prediction: predict, for instance, whether or not it will rain tomorrow

(In this last case, we most likely would actually be more interested in estimating the *probability* of rain tomorrow.)

Although much of what we will talk about will be about classification problems, there are other important learning problems. In classification, we want to categorize objects into fixed categories. In regression, on the other hand, we are trying to predict a real value. For instance, we may wish to predict *how much* it will rain tomorrow. Or, we might want to predict how much a house will sell for.

A richer learning scenario is one in which the goal is actually to behave intelligently, or to make intelligent decisions. For instance, a robot needs to learn to navigate through its environment without colliding with anything. To use machine learning to make money on the stock market, we might treat investment as a classification problem (will the stock go up or down) or a regression problem (how much will the stock go up), or, dispensing with these intermediate goals, we might want the computer to learn directly how to decide to make investments so as to maximize wealth.

3 Goals of Machine Learning Research

The primary goal of machine learning research is to develop general purpose algorithms of practical value. Such algorithms should be efficient. As usual, as computer scientists, we care about time and space efficiency. But in the context of learning, we also care a great deal about another precious resource, namely, the amount of data that is required by the learning algorithm.

Learning algorithms should also be as general purpose as possible. We are looking for algorithms that can be easily applied to a broad class of learning problems, such as those listed above.

Of course, we want the result of learning to be a prediction rule that is as accurate as possible in the predictions that it makes.

Occasionally, we may also be interested in the interpretability of the prediction rules produced by learning. In other words, in some contexts (such as medical diagnosis), we want the computer to find prediction rules that are easily understandable by human experts.

As mentioned above, machine learning can be thought of as "programming by example." What is the advantage of machine learning over direct programming? First, the results of using machine learning are often more accurate than what can be created through direct programming. The reason is that machine learning algorithms are data driven, and are able to examine large amounts of data. On the other hand, a human expert is likely to be guided by imprecise impressions or perhaps an examination of only a relatively small number of examples.

Also, humans often have trouble expressing what they know, but have no difficulty labeling items. For instance, it is easy for all of us to label images of letters by the character

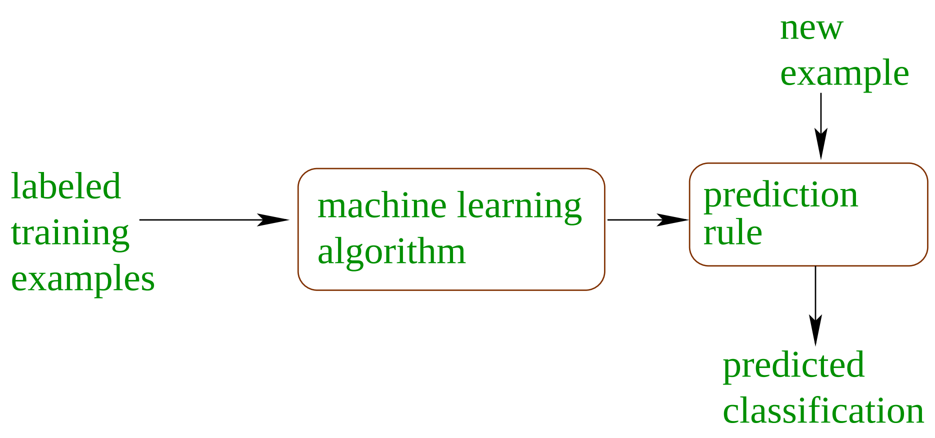

Figure 1: Diagram of a typical learning problem.

represented, but we would have a great deal of trouble explaining how we do it in precise terms.

Another reason to study machine learning is the hope that it will provide insights into the general phenomenon of learning. Some of the questions that might be answered include:

- What are the intrinsic properties of a given learning problem that make it hard or easy to solve?
- How much do you need to know ahead of time about what is being learned in order to be able to learn it effectively?
- Why are "simpler" hypotheses better?

This course is focused on *theoretical* aspects of machine learning. Theoretical machine learning has much the same goals. We still are interested in designing machine learning algorithms, but we hope to analyze them mathematically to understand their efficiency. It is hoped that theoretical study will provide insights and intuitions, if not concrete algorithms, that will be helpful in designing practical algorithms. Through theory, we hope to understand the intrinsic difficulty of a given learning problem. And we also attempt to explain phenomena observed in actual experiments with learning algorithms.

4 A Typical Learning Problem

In a typical learning problem, such as spam filtering, our goal is to create a spam filter. But rather than attacking the problem directly, we start by gathering a large collection of examples of email messages which are labeled as spam or non-spam. We then feed these examples to a general purpose learning algorithm which in turn produces a prediction rule capable (we hope) of classifying new email messages. The entire process is depicted in Figure 1.

To get a feeling for learning, we looked first at the learning problem shown in Figure 2. Here, examples are labeled positive ("+") or negative ("−"). In this case, the pattern is that all examples between 67 and 173 are positive, while those below 47 or above 197 are negative. Thus, test examples 111 and 23 are positive and negative, respectively, but we can only guess the correct label of 55.

228	−
67	+
138	+
209	−
156	+
46	−
197	−
6	−
173	+
111	?
23	?
55	?

Figure 2: A tiny learning problem. The line separates training and test examples.

197	+	11000101
128	−	10000000
30	−	00011110
72	−	01001000
133	−	10000101
109	+	01101101
213	+	11010101
84	+	01010100
3	−	00000011
200	?	11001000
68	?	01000100

Figure 3: A second toy learning problem. Examples were intially presented in decimal as in the left column, but later rewritten in binary as in the right column.

The second example is shown in Figure 3. Initially, examples were presented as integers, and the problem seemed difficult. Although someone suggested that the hidden pattern might be the union of two intervals, this turns out to give wrong predictions on the test examples. When the integers were written out in binary, the pattern became more evident, namely, that an example is positive if and only if the second and sixth bits (counting from the left) are 1.

Some things to notice about this experiment:

- Trying to find a rule that is consistent with the data seems natural and intuitive. Moreover, intuitively, the best rule should be the one that is simplest, although simplicity is not always so easy to define.

- It is very helpful to know something about what is being learned. Changing representations in the second problem gave a strong clue about what kind of rule to look for.

All of these issues will come up again very soon.

5 Learning Models

To study machine learning mathematically, we need to formally define the learning problem. This precise definition is called a *learning model*. A learning model should be rich enough to capture important aspects of real learning problems, but simple enough to study the problem mathematically. As with any mathematical model, simplifying assumptions are unavoidable.

A learning model should answer several questions:

- What is being learned?
- How is the data being generated? In other words, where does it come from?
- How is the data presented to the learner? For instance, does the learner see all the data at once, or only one example at a time?
- What is the goal of learning in this model?

6 Definitions

Before getting to our first learning model, we will need some definitions. An *example* (sometimes also called an *instance*) is the object that is being classified. For instance, in spam filtering, the email messages are the examples.

Usually, an example is described by a set of *attributes*, also known as *features* or *variables*. For instance, in medical diagnosis, a patient might be described by attributes such as gender, age, weight, blood pressure, body temperature, etc.

The *label* is the category that we are trying to predict. For instance, in spam filtering, the possible labels are "spam" and "not spam". During training, the learning algorithm is supplied with *labeled examples*, while during testing, only *unlabeled examples* are provided.

To make things as simple as possible, we will often assume that only two labels are possible that we might as well call 0 and 1. We also will make the simplifying assumption that there is a mapping from examples to labels. This mapping is called a *concept*. Thus, a concept is a function of the form $c : X \to \{0, 1\}$ where X is the space of all possible examples called the *domain* or *instance space*. A collection of concepts is called a *concept class*. We will often assume that the examples have been labeled by an unknown concept from a *known* concept class.

7 The Consistency Model

Our first learning model, called the consistency model, is rather unrealistic, but it is intuitive, and it is a good place to start. Many of the ideas that come up will also be of value later. The model captures the intuitive idea that the prediction rule that is derived from a set of examples should be consistent with their observed labelings.

We say that a concept class $\mathcal{C}$ is learnable in the consistency model if there is an algorithm A which, when given any set of labeled examples $(x_1, y_1), \ldots, (x_m, y_m)$, where $x_i \in X$ and $y_i \in \{0, 1\}$, finds a concept $c \in \mathcal{C}$ that is consistent with the examples (so that $c(x_i) = y_i$ for all i), or says (correctly) that there is no such concept.

Some examples:

Let $X = \{0,1\}^n$, and let $\mathcal{C}$ be the set of all monotone conjunctions on n boolean variables, such as $x_2 \wedge x_5 \wedge x_7$. (Monotone means that none of the variables are negated.) To learn this class in the consistency model, we take the bitwise AND of all of the positive examples, then form a conjunction of all variables corresponding to the bits that are still on. For instance, for the data in Figure 3, the bitwise AND of the positive examples gives 01000100, i.e., the conjunction $x_2 \wedge x_6$. This procedure clearly gives a conjunction c that is consistent with the positive examples. Moreover, by construction, any other conjunction c' that is consistent with the positive examples must contain a subset of the variables in c, so if c' is consistent with all the examples (both positive and negative), then c must be as well.

For monotone disjunctions, we can make a symmetric argument. Or we can use DeMorgan's Law $x_1 \vee x_2 = \overline{\overline{x}_1 \wedge \overline{x}_2}$ to reduce to the previous case. That is, we replace all of the variables with their negation, and reverse all of the labels, then apply the algorithm above for monotone conjunctions.

For non-monotone conjunctions, we can reduce to the monotone case by adding n new variables z_i where z_i is always set equal to $\overline{x}_i$.

A formula is in *disjunctive normal form (DNF)* if it is written as a disjunction of conjunctions, such as $(x_1 \wedge x_3) \vee (\overline{x}_2 \wedge x_7 \wedge x_9) \vee (x_2 \wedge \overline{x}_3 \wedge \overline{x}_4) \vee \overline{x}_1$ The conjunctions (such as $x_1 \wedge x_3$ in this case) are called *terms.* A formula is in *conjunctive normal form (CNF)* if it is written as a conjunction of disjunctions. The disjunctions are *clauses.* A *k-DNF* formula is a DNF formula with any number of terms but in which each term includes at most k literals (where a literal is either a variable or its negation). The example above is a 3-DNF formula. Likewise, a *k-CNF* formula is one in which each clause includes at most k literals. A *k-term DNF* is a DNF formula containing at most k terms (each one of which may include any number of literals). The example above is a 4-term DNF. Likewise, a *k-clause CNF* consists of at most k clauses.

For k a small constant, we can use a similar trick of adding new variables to learn k-CNF formulas in the consistency model. In particular, for every possible k-tuple of literals, we add one new variable whose value is always set equal to the OR of the k literals. This adds only $O(n^k)$ new variables.

Learning k-term DNF, on the other hand, turns out to be NP-complete for any constant $k \geq 2$. On the other hand, by "multiplying out" the formula, one can show that any k-term DNF can be rewritten as a k-CNF, which we just argued can be efficiently learned in the consistency model. So this is an odd situation in which one class, k-term DNF, cannot be learned in the model even though the superset class, k-CNF, can be.

We can also think about learning more geometrical classes in the consistency model. Let $X = \mathbb{R}^2$, and consider the class of axis-aligned rectangles in the plane.[1] Here, we can easily find a consistent rectangle by choosing the one defined by the largest and smallest positive example in each dimension. This will give the smallest rectangle containing all of the positive examples, and since it is the smallest one, it will be consistent with all of the negative examples unless there is no consistent rectangle. This argument generalizes immediately to hyper-rectangles in n dimensions.

Let $X = \mathbb{R}^n$, and let $\mathcal{C}$ be the class of linear threshold functions, i.e., functions that classify points as positive if and only if they lie above some defining hyperplane. That is, these functions are defined by a vector $\mathbf{w}$ and scalar b. Examples $\mathbf{x}$ are positive if and only

[1]We will often identify concepts with the set of all examples for which the concept is 1. Thus, when we refer to a rectangle as a concept, we really mean the concept (boolean function) that is 1 on points inside the rectangle and 0 on all others.

if $\mathbf{w} \cdot \mathbf{x} \geq b$. Given labeled examples $(\mathbf{x}_i, y_i)$, the consistency problem is to find $\mathbf{w}$ and b such that $\mathbf{w} \cdot \mathbf{x}_i \geq b$ if $y_i = 1$ and $\mathbf{w} \cdot \mathbf{x}_i < b$ otherwise. This is a linear programming problem that can be solved efficiently. However, this approach is virtually never followed in practice. We will return to this class several times in this course, and will learn about a number of techniques that are better from a learning perspective.

Finally, returning to the boolean domain $X = \{0,1\}^n$, suppose that $\mathcal{C}$ is the class of all DNF formulas. Finding a DNF formula consistent with a set of examples is trivial since we can define a term that exactly matches each positive example and take their disjunction. It should seem bothersome that "learning" such a rich class should be so easy. It is hard to believe that any kind of learning is happening. Indeed, this DNF will be positive on all of the positive examples in the training set, but negative on all other examples in the entire space, so in general, it will give very poor predictions on points not observed during train (in fact, it will predict that they are all negative).

So we see that the consistency model has some serious defects. Most importantly, there is no direct connection to learning. We only have an intuitive feeling that finding a rule that matches the training data should be a good way to learn. The model does not provide any direct link to any notion of learning or generalization.