COS 511: Theoretical Machine Learning

Lecturer: Rob Schapire
Scribe: Scott Yak

Lecture #2
February 6, 2014

1 Review of definitions

We are now mostly concerned with classification problems. For a particular classification problem, we have each *example/instance* coming from X, a much larger space of all possible examples called the *domain space* or *instance space*. Each example is associated with a label, which we denote as y. For simplicity, we assume there are only two possible labels.

1.1 The Concept and Concept class

A *concept*, which is what we are trying to learn, is an unknown function that assigns labels to examples. It takes the form $c : X \to \{0, 1\}$.

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.

1.2 The Consistency Model

We say that a concept class $\mathcal{C}$ is learnable in the consistency model if there is an algorithm[1] 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.

In the following sections, we'll look at some concept classes, and see if they are learnable.

2 Examples from Boolean Logic

For this section, we consider the instance space of n-bit vectors:

$$X = \{0, 1\}^n, \quad \mathbf{x} = (x_1, \ldots, x_n)$$

Each n-bit vector represents the values of n boolean variables. Examples of boolean variables could be the following:

$$\begin{aligned} x_1 &= \text{“is a bird”} \\ x_2 &= \text{“is a mammal”} \\ x_3 &= \text{“lives on land”} \\ &\vdots \\ &\textit{etc.} \end{aligned}$$

[1] One caveat is that this algorithm should be reasonably “efficient”, and by “efficient” we mean something like having a polynomial runtime bound. This caveat is important, because if we do not require the algorithm to be efficient, then we can always come up with an algorithm that tries every possible concept in the concept class, and we would be able to make the uninteresting claim that all finite concept classes are learnable, making this learning model uninteresting. In any case, we are mostly interested in learning algorithms that are efficient enough to be practical.

2.1 Monotone conjunctions

Suppose our concept class is the set of all *monotone conjunctions*. A monotone conjunction is the AND of some of our boolean variables, where we do not allow the boolean variables to be negated. One possible concept in $\mathcal{C}$ would be $c(\mathbf{x}) = x_2 \wedge x_5 \wedge x_7$, which is a function that takes in an n-bit vector and outputs either 0 (or $-$) or 1 (or $+$).

example	**label**
0 1 1 0 1	$+$
1 1 0 1 1	$+$
0 0 1 0 1	$-$
1 1 0 0 1	$+$
1 1 0 0 0	$-$

Given the labelled examples above, what is the monotone conjunction consistent with it? In this small example, the answer is $x_2 \wedge x_5$. But is there a general algorithm for finding a consistent monotone conjunction?

Consider this algorithm: We start off by only considering the positive examples, and we keep only the columns where all the entries are '1'. For example, only the second column and the fifth column are all ones, so our monotone conjunction is $x_2 \wedge x_5$.

example	**label**
0 1 1 0 1	$+$
1 1 0 1 1	$+$
1 1 0 0 1	$+$
0 1 0 0 1	

We then take this monotone conjunction and evaluate it with our negative examples. For example, $x_2 \wedge x_5$ evaluated on 00101, one of our negative examples, is $0 \wedge 1 = 0$. If our monotone conjunction evaluates to 0 on all negative examples, then the algorithm outputs that as the concept; otherwise the algorithm says that no consistent concept exists.

Is this algorithm correct? We know that any concept the algorithm outputs must be consistent because it has already been checked against the examples, but when the algorithm says that there is no consistent concept, how can we be sure that it is really the case?

We note that any monotone conjunction that is consistent with all positive examples can only contain the boolean variables output by the algorithm. This is because all other boolean variables were rejected for taking the value of 0 for at least one positive example, so if we included any of those boolean variables, the concept would wrongly label that positive example as a negative example. In the above example, we could not have included x_1 into the conjunction, since including x_1 would cause the concept to mislabel the positive example "01101" as negative. For similar reasons, we could not have included x_3 or x_4 either. This means that only subsets of the boolean variables that the algorithm selects have monotone conjunctions that are consistent with all positive examples, in this case, $\{\}, \{x_2\}, \{x_5\}$, and $\{x_2, x_5\}$.

Now that we have all possible concepts that are consistent with all positive examples, we would like to choose one that is also consistent with all the negative examples, if possible. Note that if any negative example were consistent with the monotone conjunction of any subset of the selected boolean variables, it would also be consistent with the monotone conjunction of all the selected boolean variables. For instance, any bit-vector that gets

evaluated to '0' with x_2 must also get evaluated to '0' with $x_2 \wedge x_5$. This implies that if the monotone conjunction of all the selected boolean variables is not consistent with any of the negative examples, then none of the monotone conjunctions that are consistent with all the positive examples would be consistent with that negative example either, so no consistent concept exists. In other words, if a consistent concept exists, the algorithm would have considered it, so it would only state that there are no consistent concepts if there really aren't any.

Since we have a learning algorithm for the concept class of monotone conjunctions, we can conclude that the concept class of monotone conjunctions is learnable.

2.2 Monotone disjunctions

Now let's consider the concept class of monotone disjunctions. A monotone disjunction is the OR of some of our boolean variables, where we do not allow the boolean variables to be negated. One possible concept in $\mathcal{C}$ would be $c(\mathbf{x}) = x_2 \vee x_5 \vee x_{20}$. Is this concept class learnable?

Consider the same set of examples again. Using De Morgan's rule, $\overline{x_1 \vee x_2} = \overline{x}_1 \wedge \overline{x}_2$, we can convert the examples into the monotone conjunction form by negating both the example bit-vector and the labels:

example	**label**		**example**	**label**
0 1 1 0 1	+		1 0 0 1 0	−
1 1 0 1 1	+	$\longrightarrow$	0 0 1 0 0	−
0 0 1 0 1	−		1 1 0 1 0	+
1 1 0 0 1	+		0 0 1 1 0	−
1 1 0 0 0	−		0 0 1 1 1	+

Now we just need to find the consistent monotone *conjunction* for the converted examples, and flip the conjunctions to disjunctions to obtain the consistent monotone disjunction for the original examples. This means that we have reduced the problem of learning from the monotone disjunction concept class to the already solved problem of learning from the monotone conjunction concept class, so the monotone disjunction concept class is also learnable.

2.3 Conjunctions

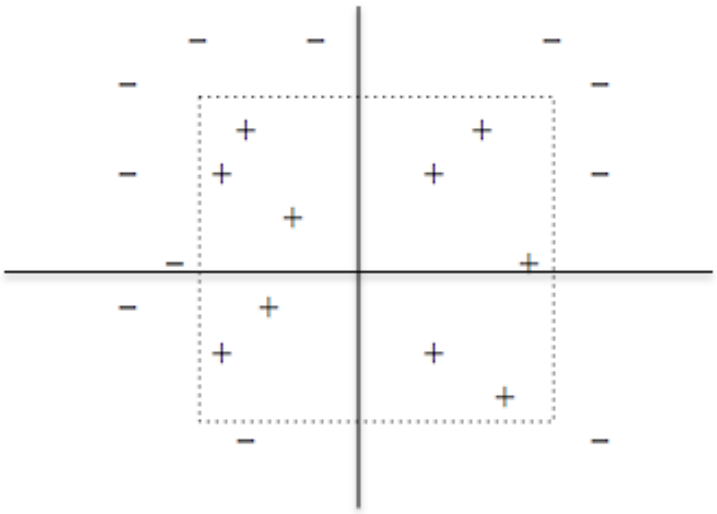
Now let the concept class be the set of conjunctions, not necessarily monotone. This means that we allow the boolean variables to be negated. For example, one possible conjunction would be $x_3 \wedge \overline{x}_7 \wedge x_{10}$, where x_7 is the negated variable. Is this concept class learnable?

Once again, we observe that this problem can also be reduced to the monotone conjunction problem. For each variable x_i in the bit-vector, we add a new variable $z_i = \overline{x}_i$, and we tack it on to the examples' bit-vectors, so $(x_1, \ldots, x_m) \rightarrow (x_1, \ldots, x_m, z_1, \ldots, z_m)$. We obtain the consistent monotone conjunction for this converted set of examples. Now suppose we obtain the monotone conjunction $x_3 \wedge z_7 \wedge x_{10}$; we would be able to convert it back to a general conjunction, $x_3 \wedge \overline{x}_7 \wedge x_{10}$. Thus, the concept class of conjunctions is also learnable.

3 Examples from Geometry

3.1 Axis-aligned rectangles

Now let's consider points on a 2-D plane, $X = \mathbb{R}^2$. Each point is assigned a label + or −. We consider the concept class of axis-aligned rectangles — each concept is a boolean function where the points in the rectangle are labelled '+', and the points outside of the rectangle are labelled '−'. So the learning problem is this: Given a set of labelled points on a plane, find an axis-aligned rectangle such that all the points inside are '+' and all the points outside are '−' (such as the dotted-line box in the following diagram).

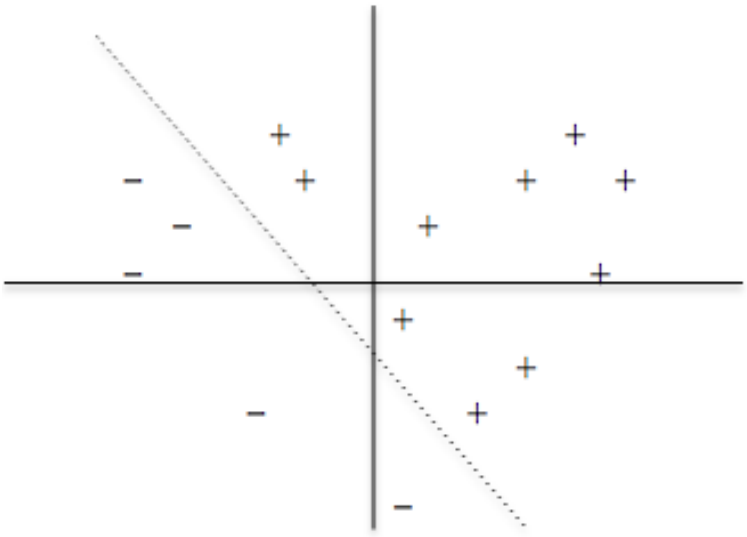


One algorithm would be to scan through all the positive examples, and use the topmost, bottommost, leftmost, and rightmost coordinates to define the smallest enclosing rectangle for the positive examples. Then we check if any of the negative examples are contained in this rectangle — if there is, we state there are no consistent concepts; otherwise we output that rectangle as the consistent concept [2]. So the concept class of axis-aligned rectangles is learnable.

3.2 Half-spaces

Let's consider points in an n-dimensional space, $X = \mathbb{R}^n$. Again, each point is assigned a label + or −. We consider the concept class of half-spaces or linear threshold functions — each concept is a boolean function where the points on one side of a linear hyperplane are labelled '+', and the points on the other side are labelled '−'. The concept we want to find is a linear hyperplane that separates the positive from the negative examples (such as the dotted-line in the diagram on the next page).

[2]Notice the general trick of finding the concept that includes as little of the domain space as possible while still remaining consistent. This trick was previously used for the monotone conjunctions



We can define a hyperplane as

$$\mathbf{w} \cdot \mathbf{x} = b$$

where $\mathbf{w}$ and b are fixed values that determine the hyperplane. We denote $\mathbf{x}_i$ as the i^{th} example, and y_i as the corresponding label. Algebraically, what we are trying to do is:

$$\begin{aligned}
&\text{find} \\
&\qquad \mathbf{w} \in \mathbb{R}^n, b \in \mathbb{R} \\
&\text{such that } \forall\ i, \\
&\qquad \mathbf{w} \cdot \mathbf{x}_i > b \text{ if } y_i = 1 \\
&\qquad \mathbf{w} \cdot \mathbf{x}_i < b \text{ if } y_i = 0
\end{aligned}$$

This turns out to be a linear programming problem, which is known to be efficiently solvable (there are math packages that can solve such linear programs well). This shows that the concept class of half-spaces is learnable.

4 More examples from Boolean logic

Once again, our domain space is the set of n-bit vectors.

4.1 k-CNF

A formula in the *conjunctive normal form*(CNF) is the conjunction of disjunctions, or the AND of ORs. For example, the following formula is a CNF:

$$(x_1 \vee x_3) \wedge (x_4 \vee \overline{x}_2) \wedge (x_2 \vee \overline{x}_3 \vee x_6) \wedge (x_7)$$

It contains *literals*, which are boolean variables either in their negated or unnegated form, and consists of *clauses* (literals ORed together) which are ANDed together. For a k-CNF, each clause can contain at most k literals, and there are no restrictions on the number of clauses in the formula. k is considered a constant. The example above would be a 3-CNF, because the longest clause contains 3 literals.

Now consider the concept class of k-CNFs. Is there an algorithm that can come up with a consistent concept? [3]

For simplicity, let's first consider 2-CNFs. One way to find a consistent concept would be to list all the possible pairs of literals (which include both the negated and unnegated forms of the boolean variables), name the OR of that as new variables, and tack them to the examples' bit-vectors (similar to what we did before with non-monotone conjunctions). This converts the CNF into a monotone conjunction[4], which we know how to solve, and once we get the solution, we can convert it back to a 2-CNF by substituting back the pairs that were named as variables. For the more general k-CNF, our new variables would include the disjunction of all subsets of the set of literals up to size k. Thus, the concept class of k-CNF is also learnable.

4.2 2-term DNF

A formula in the *disjunctive normal form*(DNF) is a disjunction of conjunctions, or an OR of ANDs. It is made up of a conjunction of *terms*, which is a conjunction of an unrestricted number of literals - the size of each term can be as large as desired. Note the difference from CNF — here, it's the number of "parentheses", and not the number of things in the parentheses, that is restricted. A k-term DNF, as the name suggests, can only contain up to k-terms. For instance, $(x_1 \wedge x_3 \wedge x_4) \vee (\overline{x}_2 \wedge \overline{x}_3 \wedge x_6 \wedge x_7)$ is a 2-term DNF.

Now we consider the concept class of 2-term DNFs. Is this class learnable?

We note the distributive property of conjunction over disjunction — we can "expand" the 2-term DNF in a way analogous to how we might multiply two polynomials, but we treat the ORs like multiplication signs, and the ANDs like additions. So,

$$(x_1 \wedge x_2 \wedge x_3) \vee (\overline{x}_2 \wedge x_4) = (x_1 \vee \overline{x}_2) \wedge (x_1 \vee x_4) \wedge (x_2 \vee \overline{x}_2) \wedge (x_2 \vee x_4) \wedge (x_3 \vee \overline{x}_2) \wedge (x_3 \vee x_4)$$

This means that we can convert any 2-term DNF to a 2-CNF. Great! Does it mean that we can reduce 2-term DNF problem to 2-CNF? We could use the 2-CNF algorithm to produce a consistent 2-CNF, but there is no guarantee that it would yield a 2-CNF that can to factorized into a 2-term DNF! The fact that we can convert any 2-term DNF to a 2-CNF only shows that the 2-term DNFs form a concept class that is a subset of the concept class of 2-CNFs. Even if we find a consistent 2-CNF, it doesn't mean that we get a consistent 2-term DNF.

In fact, the 2-term DNF learning problem is NP-hard, so an efficient learning algorithm is unlikely to exist. This is an example of a concept class that is not learnable.

4.3 General DNF

Now we consider the concept class of general DNFs, where there are no restrictions on the number of terms in the DNF. Is this learnable?

Consider the following algorithm: For each positive example, we take the negated form of the boolean variable if the value is zero, or taking the unnegated form of the boolean variable if the value is 1, and AND all these literals to form a term. For example:

[3]Note that this is different from the CNF satisfaction problem. In the CNF satisfaction problem, we start with a known CNF and try to come up with a positive example. In the learning problem here, we start off with a bunch of examples, and we are trying to find the unknown CNF that is consistent with all the examples.

[4]We might get unnecessary terms such as $x_1 \vee \overline{x}_1$, but it doesn't matter because we can just set it to 0.

example	**label**	DNF term
0 1 1 0 1	+	$\overline{x}_1 \wedge x_2 \wedge x_3 \wedge \overline{x}_4 \wedge x_5$
1 1 0 1 1	+	$x_1 \wedge x_2 \wedge \overline{x}_3 \wedge x_4 \wedge x_5$
0 0 1 0 1	−	-

Once we obtain the terms for all the positive examples, we OR them together to form a DNF. This DNF is automatically consistent with all the positive examples by construction. We then take this DNF and check if it is consistent with the negative examples, and if it is, the algorithm outputs the DNF; otherwise it outputs "no consistent DNF". Since this algorithm works, the general DNF concept class is learnable.

However, this algorithm seems to be performing a somewhat unsatisfactory kind of "learning". For one, the DNF that this algorithm outputs is going to label any instance it has not seen before as negative! Also, the DNF that the algorithm outputs is of a size comparable to the size of the training data. So in effect, this "learning" algorithm is really nothing more than a lookup table.

5 Problems with the Consistency model

One problem we saw with the DNF example is that the consistency model doesn't say anything about how the concept that the algorithm learns generalizes to new data. Even though the concept class is apparently learnable under the consistency model, it does so in an unsatisfactory way that seems unrelated to what we typically mean when we say "learning". It seems, then, that the consistency model doesn't really say much about learning at all! This suggests that we need a new model.

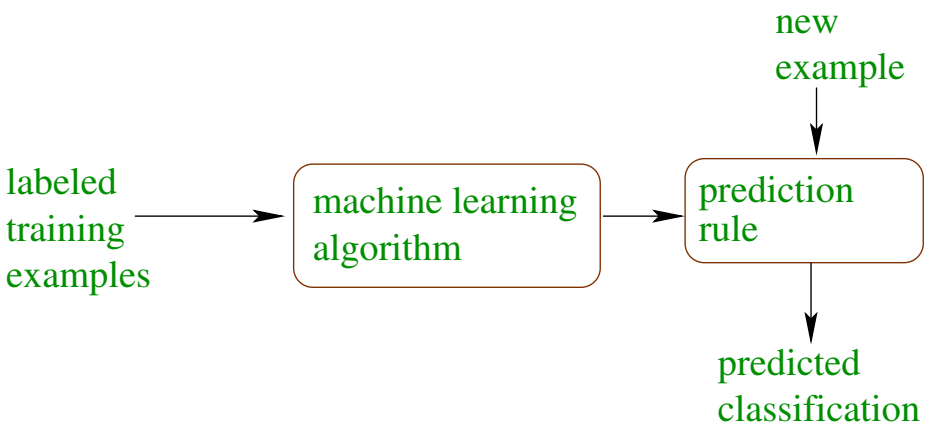
6 Quick review of probability concepts

event	—	Something that either happens or not. A probabilistic outcome.
random variable	—	Variable that takes values probabilistically. Has a distribution.
distribution	—	$Pr[X = x]$, and $\sum_x Pr[X = x] = 1$
expected value	—	$E[X] = \sum_x Pr[X = x] \cdot x$ $E[f(X)] = \sum_x Pr[X = x] \cdot f(x)$
linearity of expectations	—	$E[X + Y] = E[X] + E[Y]$. $E[cX] = cE[X]$. Always.
conditional probability	—	$Pr[a\|b] = \frac{Pr[a \wedge b]}{Pr[b]}$
independence	—	a, b are independent if $Pr[a \wedge b] = Pr[a] \cdot Pr[b]$ $\forall$r.v.$A, B, \forall a, b : Pr[A = a \wedge B = b] = Pr[A = a] \cdot Pr[B = b]$ $(shorthand : Pr[A \wedge B] = Pr[A] \cdot Pr[B])$
If X, Y independent,	$\Rightarrow$	$E[XY] = E[X] \cdot E[Y]$
union bound	—	$Pr[a \vee b] \leq Pr[a] + Pr[b]$

7 Introduction to the PAC learning model

We aim to capture some notion of learning in this new model. This means that this model should be able to say something about generalizing from a smaller set of data to a larger

set of instances.



Our goal now is to develop a hypothesis that is as accurate as possible — we are not as concerned about discovering the underlying truth. An important question we need to consider is — where do the examples come from? We need to assume that the examples are in some way related — otherwise we cannot learn anything. In this model, we would make the assumption that the examples are generated randomly from the same distribution (not necessarily uniform).

Since our goal is to obtain an accurate hypothesis, we need to be precise about what we mean by "accurate". We define the generalization/true error as follows:

$$Pr_{x\sim D}[h(x) \neq c(x)] = err_D(h)$$

where x is some example that comes from an unknown target distribution D that generates each example, and h is a hypothesis. We typically assume that the different examples are independent of each other (even though that is not always realistic, for instance, in the case of an email spam filter). We also assume that the training examples and test examples come from the same source — i.e., they are drawn from the same distribution D.

As for the labels, we assume for simplicity that there are only two possible values. The labels are applied in accordance to some unknown concept, $c : X \to \{0, 1\}$, that comes from a known concept class $\mathcal{C}$.

As an overview:

$$\begin{array}{ccccc} \text{Labelled} & & & & \text{test example } x \\ \text{training} & & & & x \sim D \\ \text{examples} & & & & \downarrow \\ (x_1, c(x_1)) & & & & \\ \vdots & & \text{Learning algorithm} & & \text{Hypothesis} \\ (x_m, c(x_m)) & \longrightarrow & \text{A} & \longrightarrow & \text{H} \\ & & & & \downarrow \\ x_i \sim D & & & & \text{label } h(x) \end{array}$$

We reiterate that in this learning model, we aim to formulate a hypothesis that has $err_D(h)$ that is as small as possible. We no longer view the goal of the learning algorithm as that of obtaining a hypothesis that is consistent on the training set. Rather, the goal now is to obtain a hypothesis that does well on test data, and that is at least "approximately correct". But since the training examples are considered to be drawn randomly from an

unknown distribution, there is always a chance that the training set that is drawn is very unrepresentative of the source distribution. For instance, there is a non-zero probability that an OCR program never sees the letter 't' up till a time we test it on a normal piece of text. Unlikely, but possible. For us to be able to say anything useful about the hypothesis, then, we would need to disregard extremely unlikely events that can completely mess up the machine learning algorithm. Thus, the accurate guarantees are "probabilistic". This sets us up for the "Probably Approximately Correct" (PAC) learning model.

Next time, we will elaborate more on the PAC learning model.