COS 511: Theoretical Machine Learning

Lecturer: Rob Schapire
Scribe: Cathy Chen

Lecture #2
February 07, 2018

1 Examples of Learning with the Consistency Model

We start by looking at some examples of algorithms that can find a consistent concept from a specified concept class that is consistent with a given dataset (or reports that no such concept exists).

1.1 Monotone Conjunctions

Problem: In our first example, we assume that the domain is $X = \{0,1\}^n$. In this case, each example is a bit vector $\mathbf{x} = (x_1, ..., x_n)$ where $x_i \in \{0,1\}$. Our concept class is the set of monotone conjunctions. This is the set of functions defined as the AND of some variables, none of which are negated. For instance, $c(\mathbf{x}) = x_2 \wedge x_3 \wedge x_7$ is a valid monotone conjunction.

Algorithm: We could use the following algorithm to solve this problem. We consider all the positive examples, and find the indices containing 1 for each training example. We propose the concept consisting of the AND of these indices. If this concept returns 0 for each negative example, we return the concept. If it does not, then we state that no consistent concept exists.

Example: For instance, we might receive the following dataset.

01101	+
11011	+
11001	+
00101	-
11000	-

Since the indices x_2 and x_5 contain 1 for each positive example, we propose $x_2 \wedge x_5$ as our concept. This concept rejects each negative training example, so it is consistent with our dataset.

Note: If any consistent concept exists for this dataset in our concept class, this algorithm finds it. Any valid concept must return 1 for all positive examples and 0 for all negative examples. Since the algorithm proposes the concept that ANDs every index which is positive for all train examples, any valid concept must be a subset of the algorithm's chosen subset. Therefore if a valid concept returns 0 on all negative examples, so does the algorithm's proposed concept.

1.2 Reductions to Monotone Conjunctions

In the following examples, we consider concept classes that reduce to the class of monotone conjunctions. We continue to assume $X = \{0,1\}^n$.

1.2.1 Disjunctions

In this example, our concept class is the set of disjunctions: concepts defined as the OR of non-negated variables. Using De Morgan's law that $\overline{x_i \vee x_j} = \bar{x_i} \wedge \bar{x_j}$, we can reduce this example to finding a monotone conjunction by flipping the label and bits of each training example. For instance, $x_2 \vee x_5$ is equivalent to $\overline{\bar{x_2} \wedge \bar{x_5}}$. If $x_2 \vee x_5$ is a consistent concept in the class of disjunctions, we could find it by finding the conjunction $\overline{\bar{x_2} \wedge \bar{x_5}}$ that is consistent with the dataset formed by flipping the label and bits of each training example in the dataset.

1.2.2 Conjunctions

Our concept class is the set of all conjunctions. We reduce this to finding a monotone conjunction by forming a new dataset. We form $\mathbf{z}_i$ by negating each bit of the original example $\mathbf{x}_i$; we then concatenate $\mathbf{x}_i$ and $\mathbf{z}_i$ to form the corresponding example in the new dataset. For instance, 1101 would map to 11010010 in our new dataset.

1.2.3 k-CNF

Our concept class is the set of k-CNFs, which is the AND of ORed clauses, each of which contains at most k terms. For instance, $(x_1 \vee x_2) \wedge (x_1 \vee x_2 \vee x_3)$ is a 3-CNF consisting of two clauses. We assume that k is a small number. We reduce this to finding a monotone conjunction by creating a dummy variable for each possible conjunction.

Note: Although this algorithm is $O(n^k)$, where n is the number of bits in each example, our assumption that k is small makes it okay.

1.2.4 2-term DNF

Our concept class is the set of 2-term DNFs. These are functions defined by the OR of two clauses, in which each clause ANDs together any number of variables. This is an NP-hard problem, which is interesting because using boolean algebra we can write a 2-tern DNF as a 2-CNF. But even though we solve 1.2.3 in polynomial time, finding a 2-term DNF is NP-hard.

1.2.5 DNF

Our concept class is the set of all DNFs, which are the OR of AND clauses. To solve this we first can form a conjunction for each positive example, where the conjunction is true only for the corresponding example, and OR all these conjunctions together.

But this creates solutions that are long and not particularly elegant. And it feels too easy to count as "learning".

1.3 Set-based Concept Classes

In the previous examples, we considered concept classes consisting of functions. In the following examples, we consider concept classes consisting of sets.

1.3.1 Axis-Aligned Rectangles

We assume the domain $X = \mathbb{R}^2$, and a concept class consisting of all axis-aligned rectangles. We want to find a rectangle such that all positive examples fall inside the rectangle, and all negative examples fall outside it. To find a consistent example, we can choose the leftmost, rightmost, topmost, and bottommost positive examples and draw the smallest possible rectangle including these points. If any negative examples fall inside this rectangle, we state that no consistent concept exists.

1.3.2 Linear Threshold Functions

In this example, we assume the domain $X = \mathbb{R}^n$, and a concept class consisting of all linear threshold functions. A consistent concept $\mathbf{w} \cdot \mathbf{x} = b$ must have the property that $\mathbf{w} \cdot \mathbf{x_i} > b$ for all positive examples x_i, and $\mathbf{w} \cdot \mathbf{x_i} < b$ for all negative examples. We can formulate this as a linear program, so we could solve this problem by throwing it into a linear program solver. Other methods of solving this problem will come later.

1.4 Analysis of Consistency Model

In the previous subsections we see that this learning model does not explicitly encourage an algorithm that generalizes past the examples it sees, and our solutions don't seem to qualify as "learning". Also, the algorithms don't allow for noise in the data. This motivates our next learning model, but first we review some definitions from probability.

2 Review of Probability Terminology

An *event* is a probabilistic outcome, such as a coin flip resulting in "heads". A *random variable* is a variable that takes possible values, such as a variable representing the outcome of a die roll. A *distribution* is a function $Pr[X = x]$ such that for all x,

$$Pr[X = x] \geq 0$$

$$\sum_x Pr[X = x] = 1$$

. The *expected value* of a random variable X is defined as

$$\sum_x Pr[X = x]x$$

. Expected value has the properties that

$$E[f(X)] = \sum_x Pr[X = x]f(x)$$

$$E[X + Y] = E[X] + E[Y] \; (linearity\, of\, expectation)$$

$$E[cX] = cE[X]$$

. We define *conditional probability* as

$$Pr[a|b] = \frac{Pr[a \wedge b]}{Pr[b]}$$

. We say that a and b are *independent* if knowing about one doesn't tell you more about the other: $Pr[a|b] = Pr[a]$ (or, equivalently, $Pr[a \wedge b] = Pr[a]Pr[b]$). We say that two random variables X and Y are independent if $\forall x, y$, the events $X = x$ and $Y = y$ are independent. If this is true, then $E[XY] = E[X]E[Y]$.

3 PAC Model

As we note in section 1.4, the consistency model leaves something to be desired. Therefore we look at another learning model.

First, we define the rule outputted by a learning algorithm as the hypothesis h, and we make three assumptions. First, we assume that each training example is chosen independently at random from an unknown, fixed, arbitrary target distribution D. We assume that our test examples are selected in the same way, and we assume that the examples are labeled according to a target concept $c : X \rightarrow \{0, 1\}$ that is unknown but inside the concept class $\mathcal{C}$ we consider.

To evaluate a hypothesis h, we define the error of h with respect to a distribution D as $err_h(D) = Pr_{x\sim D}[h(x) \neq c(x)]$ which is the probability that h misclassifies a random test example $\mathbf{x}$.

We want to find a hypothesis with small error on the test set; that is, we want a hypothesis that is "approximately correct". Since our training examples are chosen at random, it's possible that we have really bad luck and the training examples are extraordinarily skewed in some way; to account for this we look for something that is "probably approximately correct", meaning that with high probability over the choice of the training set, we want the hypothesis to be approximately correct. Therefore, we look to the probably approximately correct learning model.