COS 511: Theoretical Machine Learning

Lecturer: Rob Schapire — Lecture #2
Scribe: Meryem Essaidi — February $6, 2019$

1 Learning under the Consistency Model

1.1 Quick Recap

In the previous lecture, we saw that:

- We assume that there are only two possible *labels*: 0 and 1.
- We assume that there is a mapping from examples to labels, called a *concept*.
- X is the space of all possible examples called the *domain/instance space*.
- A *concept* c is thus a binary function of the form $c : X \longrightarrow \{0, 1\}$
- A collection of concepts is called a *concept class* C
- We assume that the examples are labeled by an unknown concept from a known concept class.

1.2 Consistency Model

We start by introducing our first model: the *Consistency Model*. This model is not ideal, but it will be used as a building block.

We say that a concept class C is *learnable in the consistency model* if there is an algorithm A that takes as input any set of labeled examples $(x_1, y_1), ..., (x_m, y_m)$, where $x_i \in X$ and $y_i \in \{0, 1\}$, and outputs a concept $c \in C$ that is consistent with the examples (so that $c(x_i) = y_i$ for all i), or says that there is no such concept. We are interested in finding efficient algorithms in this model. Note that a concept class is not necessarily finite, and if it is, then its size will usually be too large for us to afford trying all concept classes to find the right one.

We now look 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).

1.3 Monotone Conjunctions

Problem: Let the domain be $X = \{0, 1\}^n$. Each instance is a bit vector $\mathbf{x} = (x_1, ..., x_n)$ with $x_i \in \{0, 1\}$. The concept class is the set of monotone conjunctions, i.e. the AND of a subset of the non-negated variables (e.g. $c(\mathbf{x}) = x_2 \wedge x_5 \wedge x_{13}$ is a valid monotone conjunction).

Solution: We propose algorithm A that looks at all the positive training examples, and ANDs the indices of all the columns that have 1 bits for all these positive examples. Let concept c consist of the AND of these indices. If c also returns 0 for all negative examples, we return c as a valid concept. If it does not, then we claim that no consistent concept exists.

Example: Consider the following dataset:
01101+
00101−
11011+
11001+
11000−

We are explicitly checking consistency on all examples, so whenever the algorithm outputs a concept that it says is consistent, it must be correct in making that assertion.

The trickier part is arguing that if there exists any "true" concept that actually is consistent, then the algorithm will find it. To show this, we note that all of the variables in the true conjunction must be 1 on all the positive examples (since it is consistent with these). In this instance, indices x_2 and x_5 are 1 for all the positive examples. We thus propose the conjunction $x_2 \wedge x_5$ as the concept.

That means that these variables must be a subset of the ones found by our algorithm, namely that the concept outputted by algorithm A is a monotone conjunction that is a *subset of the true concept's conjunction.*

Further, for each negative example, at least one variable in the true conjunction must be 0, so it must also be 0 for the conjunction found by our algorithm, which means that our conjunction will be consistent on the negative examples as well. The conjunction $x_2 \wedge x_5$ rejects all the negative training example, and is thereby consistent with the dataset.

2 Reductions

In the following examples, we study concept classes that reduce to the class of monotone conjunctions.

2.1 Monotone Disjunctions

In this example, the concept class is the set of disjunctions, i.e. the OR of non-negated variables. Using De Morgan's law, we can reduce disjunctions to monotone conjunctions by flipping the label and bits of all training examples (e.g. $x_2 \vee x_5 = \neg(\neg x_2 \wedge \neg x_5)$). If the OR formula is a consistent concept in the class of monotone disjunctions, then the equivalent AND formula is a consistent concept in the class of monotone conjuctions. Namely, it is consistent with the new dataset formed by flipping the label and bits of each training example in the original dataset.

2.2 Non-Monotone Conjunctions

Our concept class is the set of all conjunctions (e.g. $x_1 \wedge (\neg x_3) \wedge x_7$). We reduce this to finding a monotone conjunction by forming a new dataset. For each of the variables, we create a new variable z_i by negating each bit of the original variable x_i. We then concatenate the x_i's and z_i's to form the corresponding example in the new dataset (e.g. 1101 would map to 11010010 in the new dataset, and $x_1 \wedge (\neg x_3) \wedge x_7$ becomes $c(\mathbf{x}) = x_1 \wedge z_3 \wedge x_7$).

2.3 k-CNF

Our concept class is the set of k-CNFs, i.e. the conjunctions of disjunctions (each disjunction is referred to as a "clause"), where each disjunction has at most k literals (a literal is either a variable or its negation). For instance $(x_3 \vee x_4) \wedge ((\neg x_1) \vee x_5 \vee x_8)$ is a 3-CNF consisting of two clauses.

We reduce this to finding a monotone conjunction by creating a new variable for each possible conjunction of at most k literals. This algorithm is $O(n^k)$, which is the number of new variables created, where n is the number of bits in each example, and we assume that k is a small constant, thus making this polynomial time. If k is arbitrarily large, the runtime will be exponential.

3 Additional Concept Classes

In the previous section, we considered concept classes that reduce to monotone conjunctions. In the following examples, we consider additional concept classes.

3.1 Axis-Aligned Rectangles

We assume the domain $X = \mathbb{R}^2$, and a concept class consisting of all axis-aligned rectangles (i.e. the sides of the rectangle are parallel to the axis). 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 rectangle, we can choose the right-most, left-most, bottom-most and top-most positive examples, and use these to define the right, left, bottom and top edge of the rectangle. That is, we draw the smallest possible rectangle including these points. If any negative examples fall inside this rectangle, we state that no consistent concept exists. Note that the rectangle outputted by algorithm A is *contained* in the true concept's rectangle.

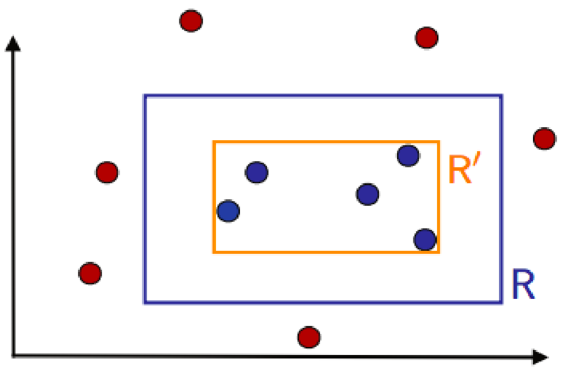

Figure 1: Target concept R and a possible consistent concept R'

In Figure 1, the target concept is R and possible consistent hypothesis is R'. Circles represent training instances. A blue circle is a point labeled with 1, since it falls within the rectangle R. Others are red and labeled with 0.

More generally, when the domain $X = \mathbb{R}^n$, and the concept class consists of all axis-aligned hyper-rectangles, for each dimension, we find the maximum value and minimum value in that dimension among all positive examples, and we use those values to define the corresponding face of the hyper-rectangle in that dimension.

3.2 Linear Threshold Functions

In this example, the domain is $X = \mathbb{R}^n$, and the concept class consists of all linear threshold functions. A consistent concept $\mathbf{w} \cdot \mathbf{x}_i = b$ must satisfy $\mathbf{w} \cdot \mathbf{x}_i > b$ for all positive examples $\mathbf{x}_i$, and $\mathbf{w} \cdot \mathbf{x}_i < b$ for all negative examples. We can formulate this as a *linear program*, that can be solved in polynomial time.

3.3 2-term DNF

Our concept class is the set of 2-term DNFs, i.e. the OR of two arbitrary length conjunctions. We can always convert any 2-term DNF to a 2-CNF (which is polynomial time).

Nonetheless, learning 2-term DNF in the consistency model is an NP-hard problem. Finding a consistent 2-CNF does not guarantee the existence of a consistent 2-term DNF. In other words, the space of 2-term DNFs is a subset of the space of 2-CNFs, which means that a concept class C that is efficiently learnable can have a subclass that is not.

Note that running the 2-CNF algorithm will output a 2-CNF that may or may not have an equivalent 2-term DNF, since not all 2-CNF have a corresponding 2-term DNF.

3.4 DNF

Our concept class is the set of all DNFs, i.e. the OR of an arbitrary number of arbitrary-length conjunctions. To solve this, we construct a clause for each positive example with a literal for each truth value of each bit: we form one conjunction that is true for each one of the positive examples, then OR all these conjunctions together.

Example: Consider the example from above:

01101+	gives	$(\neg x_1) \wedge x_2 \wedge x_3 \wedge (\neg x_4) \wedge x_5$
00101−		
11011+	gives	$x_1 \wedge x_2 \wedge (\neg x_3) \wedge x_4 \wedge x_5$
11001+	gives	$x_1 \wedge x_2 \wedge (\neg x_3) \wedge (\neg x_4) \wedge x_5$
11000−		

We now OR all these positive examples to get the following consistent concept:

$$\begin{aligned} c(\mathbf{x}) = ((\neg x_1) \wedge x_2 \wedge x_3 \wedge (\neg x_4) \wedge x_5)\\ \vee\ (x_1 \wedge x_2 \wedge (\neg x_3) \wedge x_4 \wedge x_5)\\ \vee\ (x_1 \wedge x_2 \wedge (\neg x_3) \wedge (\neg x_4) \wedge x_5)\end{aligned}$$

Constructing this DNF concept in such a way clearly works in general. This is efficient at learning a consistent DNF, but creates solutions that are too long and feel too trivial to be considered “learning”. This is similar to what we saw in the first lecture, on how the rule that lists all the positive examples in its statement (e.g. “True if it's a leopard or a zebra or a bat or a mouse”) does not generalize well.

4 PAC Model

In the previous subsections we see that this Consistency learning model does not generate algorithms that are guaranteed to succeed past the training examples, and the solutions do not seem to qualify as “learning”. The algorithms also do not allow for noise in the data and have no control over how well they generalize to new data. We now review some probability definitions to introduce our next learning model.

4.1 Probability Review

An *event* x is an outcome to which a probability is assigned. A Random Variable X is a variable that takes on values probabilisticaly.

A *distribution* is a function $Pr[X = x]$ s.t:

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

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

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

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

The following are properties of the *Expectation*:

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

$$E[X+Y] = E[X] + E[Y]$$

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

In particular, the last two properties are called *Linearity of Expectation*, which holds for all random variables, whether or not they are independent.

Conditional probability is defined as the probability of "a given b":

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

Independence is defined as follows:

- Events a and b are independent if knowing about one does not reveal anything about the other, i.e. $Pr[a|b] = Pr[a]$ which is equivalent to $Pr[a \wedge b] = Pr[a]Pr[b]$
- Random variables X and Y are independent if $\forall x, y$, the events $X = x$ and $Y = y$ are independent.

Additionaly, if X and Y are independent then $E[XY] = E[X]E[Y]$.

4.2 PAC: Probably Approximately Correct

The Consistency Model seen above is not ideal. There is nothing in the model that says how well the learner will be able to make predictions on new data, and there is no direct connection to generalization. The model also had peculiar properties. Indeed, as we saw in the 2-term DNF example above, a class can be learnable in this model even though a subclass is not. We also saw that a class like DNF is trivially learnable in the model in a way that feels like no real learning is actually happening. We now look at another learning model. We measure success as the raw accuracy of the predictions on new data.

Let *hypothesis* h be the rule outputted by a learning algorithm. We make the following assumptions:

1. The training examples are independent and identically distributed (i.i.d.) from an unknown arbitrary target distribution D.
2. The test examples are also i.i.d. from D
3. All the examples are labeled according to a target concept $c : X \longrightarrow \{0, 1\}$ that is unknown but inside the concept class C we know of .

Define the error of h with respect to a distribution D as the probability that h misclassifies a random example x from D:

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

The goal is to find a hypothesis with small error on the test set: i.e. one that is "approximately correct". In the worst case scenario, the training examples can be ridiculously skewed (which can happen because our training examples are chosen at random). Thus, we try to find a hypothesis that is “probably approximately correct”: i.e. one that is approximately correct with high probability over the choice of the training set.

Therefore, we have the *Probably Approximately Correct* (PAC) learning model.