8803 Machine Learning Theory

Maria-Florina Balcan Lecture 2: August 25, 2011

Plan: Discuss the PAC model and talk about simple PAC algorithms for learning boolean classes.

1 The PAC Model

Definition 1 *We say that algorithm $\mathcal{A}$ learns class C in the consistency model if given any set of labeled examples S, the algorithm produces a concept $c \in C$ consistent with S if one exists, and outputs "there is no consistent concept" otherwise. The algorithm runs in polynomial time (in the size of S and the size n of the examples).*

The basic idea of the PAC model is to assume that examples are being provided from a fixed (but perhaps unknown) distribution over the instance space. The assumption of a fixed distribution gives us hope that what we learn based on some training data will carry over to new test data we haven't seen yet.

Definition 2 *Given a example distribution $\mathcal{D}$, the* error *of a hypothesis h with respect to a target concept c is $\mathbf{Prob}_{x \in \mathcal{D}}[h(x) \neq c(x)]$. ($\mathbf{Prob}_{x \in \mathcal{D}}(A)$ means the probability of event A given that x is selected according to distribution $\mathcal{D}$.)*

In the following definition, "n" denotes the size of an example.

Definition 3 *An algorithm $\mathcal{A}$ PAC-learns concept class C by hypothesis class H if for any $c^* \in C$, any distribution D over the instance space, any $\epsilon, \delta > 0$, and for some polynomial p, the following is true. Algorithm $\mathcal{A}$, with access to labeled examples of c^* drawn from distribution $\mathcal{D}$ produces with probability at least $1 - \delta$ a hypothesis $h \in H$ with error at most ϵ. In addition,*

1. *$\mathcal{A}$ runs in time polynomial in n and the size of the sample*
2. *The sample has size $p(1/\epsilon, 1/\delta, n, size(c^*))$.*

The quantity ϵ is usually called the accuracy parameter and δ is called the confidence parameter. A hypothesis with error at most ϵ is often called "ϵ-good."

Note: This definition allows us to make statements such as: "the class of k-term DNF formulas is learnable by the hypothesis class of k-CNF formulas."

Remark 1: If we require $H = C$, then this is typically called "proper PAC learning". If we allow H to be the class of polynomial time programs (i.e., we don't care what representation the learner uses so long as it can predict well) then this is typically called "PAC prediction". I will usually say: "concept class C is PAC-learnable" to mean that C is learnable in the PAC-prediction sense.

Remark 2: One nice extension of this model is instead of requiring the error of h by at most ϵ to just require that the error be at most $\frac{1}{2} - 1/poly(n)$. This is called *weak learning* and we will talk more about this later.

Remark 3: Another nice extension is to the case where H is not necessarily a superset of C. In this case, let ϵ_H be the least possible error using hypotheses from H. Now, we relax the goal to having the error of h be at most $\epsilon + \epsilon_H$. If we let C be the set of all concepts (and we remove "$size(c^*)$" from the set of parameters we are allowed to be polynomial in), then this is often called the *agnostic model* of learning: we simply want to find the (approximately) best $h \in H$ we can, without any prior assumptions on the target concept.

1.1 Relating the Consistency and the PAC model

Generalizing the case of conjunctions, we can relate the Consistency and the PAC model as follows.

Theorem 1 *Let $\mathcal{A}$ be an algorithm that learns class C in the consistency model (i.e., it finds a consistent $h \in C$ whenever one exists). Then $\mathcal{A}$ needs only*

$$\frac{1}{\epsilon}\left(\ln |C| \ + \ \ln \frac{1}{\delta}\right)$$

examples to output a hypothesis of error at most ϵ with probability at least $1-\delta$. Therefore, $\mathcal{A}$ is a PAC-learning algorithm for learning C (by C) in the PAC model so long as this quantity is polynomial in $size(c)$ and n.

Note: If we learn C by H, we just need to replace $\ln |C|$ with $\ln |H|$ in the bound. For example, if $\ln |H|$ is polynomial in n (the description length of an example) and if we can find a consistent $h \in H$ in polynomial time, then we have a PAC-learning algorithm for learning the class C.

Proof: We want to bound the probability of the following bad event.

B : $\exists h \in C$ with $err_D(h) > \epsilon$ and h is consistent with S.

To do so, let us first fix a bad hypothesis, i.e., a hypothesis of error at least ϵ. The probability that this hypothesis is consistent with m examples is at most $(1-\epsilon)^m$. So, by union bound, the probability that there exists a bad hypothesis consistent with the sample S is at most $|C|(1-\epsilon)^m$.

To get the desired result, we simply set this to δ and solve for m . ■

The above quantity is polynomial for conjunctions, k-CNF, and k-DNF (for constant k). It is not polynomial for general DNF. It is currently unknown whether the class of DNF formulas is learnable in the PAC model.

Note: Theorem 1 also that this theorem requires C (or H) to be finite. We will discuss in a couple of lecture the case where C (or H) is not finite.

1.2 Examples

Assume each example x is given by n boolean features (variables).

AND functions (monotone conjunctions). We can learn this class in the consistency model by the following method:

1. Throw out any feature that is set to 0 in any positive example. Notice that these cannot possibly be in the target function. Take the AND of all that are left.
2. If the resulting conjunction is also consistent with the negative examples, produce it as output. Otherwise halt with failure.

Since we only threw out features when absolutely necessary, if the conjunction after step 1 is not consistent with the negatives, then no conjunction will be. There are at most 2^n monotone conjunctions, so by Theorem 1 the class of monotone conjunctions is learnable in the PAC model.

Non-monotone conjunctions, disjunctions, k-CNF, k-DNF. What about functions like $x_1\bar{x}_4x_7$? Instead of thinking about this from scratch, we can just perform a reduction to the monotone case. If we define $y_i = \bar{x}_i$ then we can think of the target function as a monotone conjunction over this space of $2n$ variables and use our previous algorithm. k-CNF is the class of Conjunctive Normal Form formulas in which each clause has size at most k. E.g., $x_4 \wedge (x_1 \vee x_2) \wedge (x_2 \vee \bar{x}_3)$ is a 2-CNF. So, the 3-CNF learning problem is like the inverse of the 3-SAT problem: instead of being given a formula and being asked to come up with a satisfying assignment, we are given assignments (some satisfying and some not) and are asked to come up with a formula. k-DNF is the class of Disjunctive Normal Form formulas in which each term has size at most k. We can learn these too by reduction: e.g., we can think of k-CNFs as conjunctions over a space of $O(n^k)$ variables, one for each possible clause. There are at most $2^{O(n^k)}$ k-CNFs, so for constant k, by Theorem 1 the class of k-CNFs is learnable in the PAC model.

Decision lists. A Decision List is a list of if-then rules: "if ℓ_1 then b_1, else if ℓ_2 then b_2, else if ℓ_3 then b_3, ..., else b_m", where each ℓ_i is a literal (either a variable or its negation) and each $b_i \in \{0, 1\}$. The meaning is: given an example, we look at the first left-hand-side satisfied and output the corresponding right-hand-side. For instance, one possible decision list is the rule: "if $\overline{x}_1$ then positive, else if x_5 then negative, else positive."

If you like, you can think of this as a decision tree with just one long path.

How many decision lists are there? From each variable, we can form four rules by specifying either the variable or its negation on the left-hand side, and either 0 or 1 on the right-hand side. As each rule classifies all those examples to which it applies, there is no point in applying a given rule more than once. These two observations taken together allow us to set an upper bound of $(4n)!$ on the number of decision lists. Further analysis leads us to a somewhat tighter bound of $n!4^n$. In either case, $\ln |C| = O(n \log n)$.

An algorithm for learning DLs in the consistency model.

1. Find some rule consistent with the current set of examples that applies to at least one of them. If no such rule exists, halt with failure.
2. Put the rule at the bottom of the hypothesis.
3. Throw out those examples classified by the hypothesis so far.
4. If there are any examples left, repeat from the beginning.

It is clear that this algorithm runs in polynomial time, since at any stage there are at most $4n$ rules to compare with the data, and the algorithm can run for at most $4n$ iterations. We now show that if there exists a consistent list, this algorithm will find one.

Suppose there exists some consistent decision list c^*, and suppose that some examples still remain in our data set. Let R be the highest rule in c^* that applies to at least one example in our current data set. Notice that R must be consistent with the data because, by the definition of a decision list, any inconsistent example must cause a higher rule in c^* to fire, which contradicts our definition of R. Therefore, our algorithm has at least one legal choice (namely, rule R) in step 1.

Since $\ln |C| = O(n \log n)$, by Theorem 1, the class of decision lists is learnable in the PAC model.

2 Learning Linear Separators

Here we can think of examples as being from $\{0, 1\}^n$ or from R^n. In the consistency model, the goal is to find a hyperplane $w \cdot x = w_0$ such that all positive examples are on one side and all negative examples are on other. I.e., $w \cdot x > w_0$ for positive x's and $w \cdot x < w_0$ for negative x's. We can solve this using linear programming. The sample complexity results for classes of finite VC-dimension that we will cover later in the course together with known results about linear programming imply that the class of linear separators is learnable in the PAC model. Next time, we will talk about the Perceptron algorithm, an online algorithm for learning linear separators, one of the oldest algorithms used in machine learning (from early 60s).