CS181 Lecture 8 — Computational Learning Theory

Now that we have seen a learning algorithm — decision trees — and discussed the issue of overfitting and how to deal with it, we turn to a basic study of how and when it is possible to learn, under the topic of computational learning theory. We will briefly survey some of the main ideas in this area. Les Valiant, who founded the field, teaches an in-depth course on the topic (CS228).

1 Computational Learning Theory

Learning works. There's lots of evidence of that, both about humans and about machine learning algorithms. Today we'll consider the question "Why does learning work?". There are several reasons why one might want to answer this question, including:

- Simply for the sake of understanding. To quote Russell & Norvig, "Unless we find some answers, machine learning will, at best, be puzzled by its own success."
- To understand when and under what circumstances learning works.
- To be able to give guarantees about the performance of the hypothesis

produced by an algorithm on some training set.

- To be able to determine how many training samples are needed in order to produce a good hypothesis.

Computational learning theory relies on the concept of inductive bias. As we have seen, learning is impossible without some sort of inductive bias. Intuitively, the stronger the inductive bias, the more learning will be possible. Of course, this does not come for free, because inductive bias leads to the possibility of error, and a stronger bias leads to greater possibility of error. But one can ask, assuming that one's inductive bias is actually correct, how much learning is possible? Or, to weaken the assumption, assuming that the inductive bias appears to have low error, based on looking at the training set, how much learning is possible. First of all, we need a concrete definition of a question like "how much learning is possible". The theory of PAC learning provides this, and also provides an answer to the question in some cases.

PAC learning focuses on restriction bias rather than preference bias. The main reason for this is that restriction bias is easier to characterize and analyze mathematically.

2 The PAC Framework

What does it mean to say that a learning algorithm works? An answer to this question is provided by the PAC framework (for Probably Approximately Correct), first introduced by Professor Valiant.

Why "probably approximately correct"? In general, we cannot expect a learning algorithm to produce a perfectly correct hypothesis. The best we can hope for is that it produce a hypothesis with error less than some small ϵ. Also, we cannot expect an algorithm to produce a good hypothesis for every single possible training set — some training sets may provide a very unrepresentative sample of the population. The best we can hope for is to produce an approximately correct hypothesis with probability at least $1 - \delta$ for some small δ.

In order to talk about the probability that an approximately correct hypothesis is produced, we need some probability distribution. The PAC model makes the assumption that all instances — both training and test instances — are drawn independently from the same probability distribution P. Note that this

assumption gives precise content to the statement that “the future resembles the past”, which is the fundamental assumption without which all induction is impossible.

To be precise, P is a joint probability distribution over < X,C >, the attribute values and the classification. If we want, we could view P as consisting of a distribution over attribute values P(X), and a conditional distribution over the class given the attribute values P(C | X), and the joint distribution is given by P(X,C) = P(X)P(C | X). We assume that all training and all test instances are independent samples from the distribution P.

So, given a training set size of N, P determines a distribution over the different training sets of N instances. If D is the training set consisting of instances $D_1,...,D_N$, $P(D) = \prod_{i=1}^{N} P(D_i)$. Also, P determines the expected error of some hypothesis h on test data. It is just the probability, according to distribution P, that an incorrect classification will be produced on an arbitrary instance. So we define

$$error_P(h) = P(h(x) = c).$$

Given the notion of a distribution over instances, we can make precise the definition of a deterministic vs non-deterministic domain we introduced last week. We will focus mostly on deterministic domains in the following analysis. A domain is deterministic if, for every set of attribute values x, there is a class c such that P(c | x) = 1. For a deterministic domain, we can speak of the “true model” as being the function that assigns, for each x, the class c such that P(c | x) = 1. If we fix the true model f, then to describe the distribution over instances, we need only describe P(X), since P(C | X) is fixed by f. Also,

for a deterministic domain whose true model is f, we can characterize the error of a hypothesis h as

$$error_P(h) = P(h(x) = f(x)).$$

This leads us to the following definition:

Definition 2.1: Consider a supervised learning problem with attributes $X_1,...,X_n$ and class C. A hypothesis space H is PAC-learnable if there exists an algorithm such that, for every deterministic domain whose true model f is in H, for every distribution P over $X_1,...,X_n$, and every $\epsilon,\delta > 0$, with probability at least $1 - \delta$ the algorithm returns a hypothesis $h \in H$ with $error_P(h) < \epsilon$, in time

polynomial in $1/\epsilon$, $1/\delta$ and n.

This definition is a bit of a mouthful, so let's work through it. First, it sets up the scenario - we have a supervised learning problem where we want to learn a function from $X_1 \times ... \times X_n$ to C. The parameter n represents the number of attributes. We're considering a hypothesis space H, that is some subset of the possible functions from X to C. We want to know if it is possible to learn a good hypothesis in H efficiently, assuming that the domain is deterministic, and that the true model is actually in H.[1]

The definition requires that for every possible distribution P over X, for every possible true model f in H, and for every positive ϵ and δ, the algorithm probably (with probability at least $1-\delta$) returns an approximately correct (error $< \epsilon$) hypothesis. Furthermore, the algorithm has to be efficient — it has to work in time polynomial in $1/\epsilon$, $1/\delta$ and the number of attributes n.

Implicit in the fact that the algorithm has to take polynomial time is that it only uses a polynomial number of training examples. In addition, given a sufficient number of training examples, it has to be able to produce a good hypothesis efficiently. In fact, the number of training examples needed to learn a good hypothesis is usually the main limitation on the performance of a learning algorithm. In most cases,[2] if a good hypothesis can be produced from a polynomial number of instances, it can also be produced efficiently. Since the number of training instances needed is the key limitation, we make the following definition:

Definition 2.2: The sample complexity of a hypothesis space H is the number of training instances N (as a function of n, δ and ϵ) needed to guarantee that for every distribution P over X and every true model $f \in H$, with probability at least $1 - \delta$, every hypothesis $h \in H$ consistent with a training set of size N has $\text{error}_P(h) < \epsilon$.

Why is sample complexity a useful concept? Suppose we have a learning algorithm L that has a restriction bias in that it only considers hypotheses in a hypothesis space H. Suppose we give L some training data D, and L comes back saying that hypothesis $h \in H$ is consistent with D. The sample complexity

[1] This is a strong assumption, and we'll talk about relaxing it later.

[2] We will see an exception to this later.

tells us how many training instances we need in order to guarantee that h is probably approximately correct.

So the key thing we need to know about a hypothesis space is its sample complexity. In particular, we want to know if it's polynomial in n, $1/\epsilon$ and $1/\delta$.

3 Examples

3.1 Conjunctive Formulas

Let's look at an example of a hypothesis space that we can prove has polynomial sample complexity. Consider the problem of learning boolean formulas, and let the hypothesis space H consist of conjunctive formulas. Recall that a literal has the form X_i or $\neg X_i$, and a conjunctive formula is a conjunction of literals, such as $X_1 \wedge \neg X_2$.

In order to do a PAC analysis, the first question we ask is what is the size of the hypothesis space, i.e., how many conjunctive formulas are there? The answer is 3^n, because for each attribute X_i, there are three possibilities: X_i is in the formula, $\neg X_i$ is in the formula, or neither are in the formula. Armed with the knowledge that $|H| = 3^n$, let's proceed with the analysis.

Given a number of training instances N, we're going to find a bound on the probability that there is a bad hypothesis (one with error $> \epsilon$) that is consistent with the training data. We're not going to make any assumptions about the distribution P over instances, so this bound will hold for any such distribution.

First, let h be any particular bad hypothesis. What is the probability that h is consistent with a single training instance? Clearly at most $1-\epsilon$. What is the probability that h is consistent with N training instances? At most $(1-\epsilon)^N$, since the different instances are drawn independently from the distribution P.

Now, we can ask ourselves, what is the probability that there exists some bad hypothesis consistent with a training set D of m instances? Let's enumerate the bad hypotheses $h_1,...,h_k$. The event "there is some bad hypothesis consistent with D" is equal to the event "h_1 is consistent or h_2 is consistent or ... or h_k is consistent". Now we use the union bound: the probability of the union of events is less than or equal to the sum of the probabilities of the individual events. So

$$
\begin{aligned}
&\text{P(there is some bad hypothesis consistent with D)} \\
&= \text{P}(h_1 \text{ consistent with D} \vee ... \vee h_k \text{ consistent with D}) \\
&\leq \text{P}(h_1 \text{ consistent with D}) + ... + \text{P}(h_k \text{ consistent with D}) \\
&\leq k\text{P(a bad hypothesis h is consistent with D)} \\
&< k(1-\epsilon)^N \\
&\leq 3^n(1-\epsilon)^N
\end{aligned}
$$

The last line follows since the number k of bad hypotheses is at most the size of the hypothesis space, which is 3^n.

Now that we have a bound on the probability that there is a bad hypothesis consistent with a training set of size N, we next ask what value of N is large

enough to guarantee that this probability is at most δ? I.e., we want N such that $3_n(1-\epsilon)^N < \delta$. First we use the standard inequality

$$1 - x \leq e^{-x}$$

to get $3_n e^{-\epsilon N} < \delta$. Taking logarithms and solving, we get

$$N > \frac{1}{\epsilon}(n \ln 3 + \ln \frac{1}{\delta}).$$

For such an N, we can indeed guarantee that the probability that there is a bad hypothesis consistent with the training data is at most δ. Notice that the expression for N is linear in $1/\epsilon$ and n, and logarithmic in $1/\delta$, so we conclude that conjunctive formulas have polynomial sample complexity. To give you a sense of the numbers, if n = 10, $\epsilon = 0.1$ and $\delta = 0.05$, 140 training examples suffice.

3.2 General analysis for finite hypothesis spaces

In the above analysis, we only used the fact that our hypothesis space consisted of conjunctive formulas in proving that |H| is 3_n. We can go through the exact same analysis for any finite hypothesis space, substituting |H| wherever 3_n appears. In the end, we will deduce that if

$$N > \frac{1}{\epsilon}(\ln |H| + \ln \frac{1}{\delta}),$$

then the probability that there is a bad hypothesis consistent with a training set of N samples is at most δ. In particular, if |H| is "only" exponential in n, H has polynomial sample complexity.

On the other hand, consider an unbiased hypothesis space consisting of all Boolean functions on n attributes. This hypothesis space has size 2^{2_n} — it is doubly exponential in n. In this case, plugging |H| into the formula gives

$$N > \frac{1}{\epsilon}(2_n + \ln \frac{1}{\delta}),$$

This is indication, but not a proof,[3] that H does not have polynomial sample complexity. The reason is that the formula only provides an upper bound on the number of samples needed. In particular, the union bound that we used in the analysis is a very weak bound. It holds with equality only when the different events are disjoint. The greater the overlap between events, the weaker the bound. In our case, the events "h_1 is consistent with D" and "h_2 is consistent with D" are likely to have a good deal of overlap, so the bound is a weak one.

[3] But one can indeed prove it.

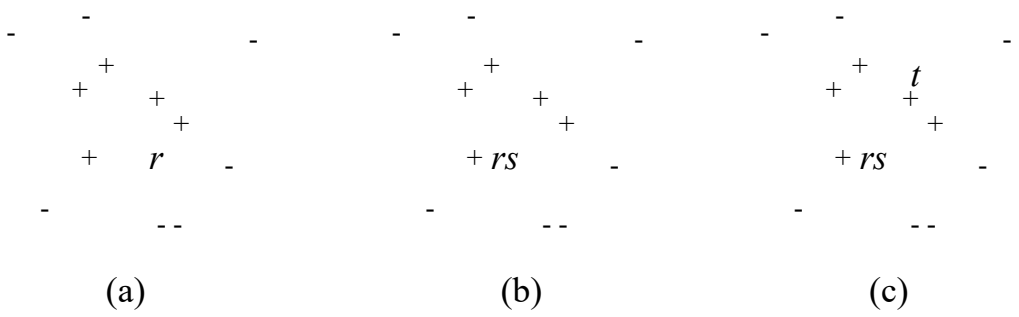

Figure 1: Circles in the plane.

3.3 k-term DNF and k-CNF formulas

The PAC-learning definition requires that with high probability a good hypothesis is returned in polynomial time. Polynomial time means both polynomial sample complexity, and that the computation can be performed efficiently.

Here's an example that highlights this point. A k-term DNF formula (DNF stands for disjunctive normal form) is the disjunction of up to k conjunctive formulas. I.e., it has the form $t_1 \vee ... \vee t_l$, where $l \leq k$ and each t_i is a conjunctive formula. One can show that k-term DNF has polynomial sample complexity. However, it turns out that k-term DNF formulas cannot be learned in polynomial time.[4]

This example illustrates the fact that polynomial sample complexity is not sufficient for PAC-learnability. Remarkably however, there is another hypothesis space that is strictly larger than k-term DNF formulae that is PAC-learnable. This is the hypothesis space of k-CNF formulae (CNF stands for conjunctive normal form). A k-CNF formula is the conjunction of any number of terms, where each term is the disjunction of up to k literals. It can be shown that every k-term DNF formula can be written as a k-CNF formula, but the converse is not true. It also can be shown that k-term CNF formulae are PAC-learnable! As this example shows, it is possible for a hypothesis space to be PAC-learnable, while some proper subset of the space is not.

3.4 An Infinite Hypothesis Space

The above analysis using the union bound only works for finite hypothesis spaces. If the attributes are continuous, the hypothesis space will usually be infinite. Nevertheless, as the following example shows, we can apply a PAC analysis with a little thought.

Here, we consider a learning problem from n continuous attributes to a Boolean classification. Our hypothesis space consists of n-dimensional spheres

[4]This is true assuming RP = NP, which is generally believed to be true. RP is the class of problems that can be solved by a randomized algorithm with high probability.

centered at the origin. Figure 1 (a) shows an example hypothesis in two dimensions. The hypothesis is characterized by a circle of radius r, and represents the points $\{(x, y) : x^2 + y^2 \leq r\}$. Also shown is a training set D consisting of positive and negative points. All positive examples lie inside the circle, while all negative examples lie outside it, so the hypothesis r is consistent with D.

Now, let us suppose that the true concept is the sphere r. Define s (shown in Figure 1 (b)) to be the smallest sphere larger than r such that $\text{error}_P(s) > \epsilon$.

Now suppose that there is some sphere t (shown in Figure 1 (c)) larger than r such that $\text{error}_P(t) > \epsilon$ and t is consistent with the training data. We know that $t \geq s$ by definition of s. Also, since t is consistent with the data, all negative examples lie outside t. In addition, since r is the true concept, all positive examples lie inside r. Since $t \geq s$ and $r \leq s$, we can conclude that all negative examples lie outside s and all positive examples lie inside s, i.e., that s is consistent with D.

We have shown that if there is any bad hypothesis larger than r consistent with D, then s is consistent with D. We can similarly define q to be the largest bad hypothesis smaller than r, and show that if there is any bad hypothesis smaller than r consistent with D, then q is consistent with D. It follows that the event “there is a bad hypothesis consistent with D” is equivalent to the event “q or s is consistent with D”.

This is all we need to show that this hypothesis space has polynomial sample complexity (and in fact it does not even depend on n). In the PAC analysis, we can use the union bound to say

$$\begin{aligned} &P(\text{there is some bad hypothesis consistent with D}) \\ &\leq P(\text{q is consistent with D}) + P(\text{s is consistent with D}) \end{aligned}$$

$$\leq 2(1-\epsilon)^m$$

So in fact, the number 2 appears where we previously used |H|. Continuing the analysis as before, we eventually see that this hypothesis space has polynomial sample complexity, even though the hypothesis space is infinite.

We'll talk about a more general method, VC dimension, for proving polynomial sample complexity for infinite hypothesis spaces after we've considered neural networks.

4 Guarantees

4.1 Consistent Learners

The PAC analyses we have done assume that the true concept is in the given hypothesis space H. This is a strong assumption. Is it possible to get useful results out of the PAC framework while relaxing this assumption? The answer is yes. First we consider a type of learning algorithm called a consistent learner. A consistent learning algorithm L with hypothesis space H is one that, given a training set D, will always return a hypothesis in H consistent with D if one exists, otherwise it will indicate that no such hypothesis exists.

Suppose H has polynomial sample complexity. The PAC model says that given ϵ, δ and n there is some number N (polynomial in $1/\epsilon$, $1/\delta$ and N) of samples such that if the true concept is in H, with probability at least $1-\delta$ an algorithm that returns a hypothesis in H consistent with the data will return a hypothesis that has error at most ϵ. With a consistent learner L we can do better. We can say that even if the true concept is not in H, if L returns a hypothesis in H saying that it is consistent with the training data, then with probability at least $1-\delta$ that hypothesis has error at most ϵ.

4.2 Agnostic Learners

We can do even better with a type of learning algorithm called an agnostic learner. An algorithm L with hypothesis space H is an agnostic learner, if, given a training set D, L returns a hypothesis in H with minimum error on D.

Let us use $\text{error}_D(h)$ for the error of hypothesis h on the training set D, and

$\text{error}_P(h)$ for the true error of h relative to the distribution P. (Recall that in the PAC model we fix a distribution from which both training and test instances are taken.) The Hoeffding bounds say that for a given h,

$$P(\text{error}_D(h) \leq \text{error}_P(h) - \epsilon) \leq e^{-2N(\epsilon 2)}.$$

It follows, using the same ideas as in the standard PAC analysis, that

$$P(\exists h \in H : \text{error}_D(h) <= \text{error}_P(h) - \epsilon) \leq |H| e^{-2N(\epsilon 2)}.$$

In particular, if h_* is the hypothesis returned by L, it follows that[5]

$$P(\text{error}_D(h_*) \leq \text{error}_P(h_*) - \epsilon) \leq |H| e^{-2N(\epsilon 2)}.$$

Continuing with the same reasoning as in the standard PAC analysis, we derive that if

$$N > \frac{1}{2\epsilon 2}(\ln |H| + \ln(\frac{1}{\delta})),$$

then with probability at least $1-\delta$, the difference between true error and training errors of the hypothesis returned by L will be at most ϵ. This is a very useful result, because it allows us to give concrete guarantees as to the performance of

[5]We can't treat h_* like a general h and use

$$P(\text{error}_D(h) \leq \text{error}_P(h) - \epsilon) \leq e^{-2N(\epsilon 2)}.$$

The reason is that the fact that h_* was returned by L is indicative that the deviation between the training and test errors of h_* might be higher than average — after all, L returns the hypothesis that has the lowest training error. Thus

$$P(\text{error}_D(h) \leq \text{error}_P(h) - \epsilon) = P(\text{error}_D(h) \leq \text{error}_P(h) - \epsilon \mid h \text{ was returned by L}).$$

However, we know for sure that

$$P(\text{error}_D(h) \leq \text{error}_P(h) - \epsilon \mid h \text{ was returned by L}) \leq P(\exists h \in H : \text{error}_D(h) <= \text{error}_P(h) - \epsilon).$$

a hypothesis on unseen data, given its performance on the training data. If L returns hypothesis h with small training error $\text{error}_D(h)$, and at least N samples have been seen, we can guarantee that with probability at least $1 - \delta$ the true error of h is no more than $\text{error}_D(h) + \epsilon$.

4.3 Overfitting to the Domain

One needs to be careful in using these guarantees about the performance of a learned hypothesis. Suppose I come to you with the following statement: "I tried lots of different learning algorithms on the domain, until I came up with one that produced a hypothesis with very low training error. Since the algorithm that returned that hypothesis has a small hypothesis space, I can use the PAC analysis to guarantee that it will with high probability also have low test error."

What's wrong? The problem is that by trying many different algorithms until I find the one that seems to do well on the data, I am implicitly using a much more complex algorithm with a much more expressive hypothesis space. If I am only going to stop searching the space of algorithms when I find one that does well (or I've exhausted all the algorithms in my repertoire), then essentially the hypothesis space of my meta-algorithm is the union of the hypothesis spaces of the individual algorithms.

Compare this meta-algorithm to a second one in which I say that I'm going to run all the individual algorithms and take the best result. Both meta-algorithms have the same hypothesis space: the union of the individual spaces. The only difference between them is that the first meta-algortithm has a preference bias for hypotheses produced by algorithms earlier in the search ordering.

This process of searching for a good algorithm that does well on the data one has leads to another overfitting issue, called overfitting to the domain. Consider the following situation. A researcher wants to apply machine learning to some practical domain. She obtains a corpus of data for the domain, and tries out various algorithms to determine which will work well. In addition to trying out different algorithms, she also adjusts parameters of the algorithms (e.g. validation set size for ID3 with validation set pruning) to determine what parameter settings work best. Being a good machine learning researcher, she divides the corpus into training data and test data, and perhaps performs cross-validation, so as to obtain accurate estimates of the performance of each learning algorithm. At the end of this process, she selects the algorithm that performed best on the test set. She is then surprised to discover that the algorithm does not perform as well in practice as it did on the test set.

What went wrong? The problem is that this whole process of looking for an algorithm that works well on the corpus is in itself a learning process. While the test data is functioning as a test set in order to measure the accuracy of the individual algorithms, it is serving as training data for the purpose of selecting an algorithm. It may be that some algorithm did better on the test set than it should have, simply due to variance in the test data. If many different algorithms are tried, it is quite likely that this will happen, though usually it will be on a small scale.