600.363 Introduction to Algorithms / 600.463 Algorithms I **Lecturer:** Michael Dinitz
Topic: Intro to Learning Theory **Date:** 12/2/14

25.1 Introduction

Today we're going to talk about machine learning, but from an algorithms point of view. Machine learning is a pretty broad and interdisciplinary field, including ideas from AI, statistics, and theoretical computer science. All of these are important ingredients, but we're going to focus on machine learning from a TCS point of view. This means in particular that we're going to focus on provable guarantees and we'll try to make our analyses as worst-case as possible, as opposed to more AI or stats-based points of view where the focus is on making probabilistic assumptions and then designing algorithms that seem to work well (but where there isn't a ton of formal proofs).

I could spend a day ranting about the lack of proofs in modern machine learning, but let's not do that. Instead, maybe I'll just note for those of you who already know a little about this that there is shockingly little justification and proofs for many standard ML algorithms/techniques such as nonnegative matrix factorization, independent component analysis, dictionary learning, etc. There is a small group of people, most notably Sanjeev Arora at Princeton, who are trying to bring some rigor back to ML (particularly the linear algebraic parts of ML). But today we're going to talk about some older stuff, from when ML and TCS were a little more tightly bound and ML people still cared about proofs.

25.2 Concept Learning

The following basic problem arises often in machine learning: there is some unknown set, and we are given a sample of some elements and told whether or not each element is in the set (i.e. we are given *labeled* data). From this data, we want to produce a good prediction rule (a *hypothesis*) for future data.

The paradigmatic example of this problem is spam categorization. We want a computer program to help us decide which emails are spam and which are important. We can assume that each email is represented by n features (e.g., return address, keywords, size, etc.). Then we are given a sample S of emails which have already been labeled as spam or not spam, and are asked to provide a rule to use in the future. For example, our input set S might look like the following.

sales	size	Mr.	bad spelling	known-sender	spam?
Y	N	Y	Y	N	Y
N	N	N	Y	Y	N
N	Y	N	N	N	Y
Y	N	N	N	Y	N
N	N	Y	N	Y	N
Y	N	N	Y	N	Y
N	N	Y	N	N	N
N	Y	N	Y	N	Y

Given this data, a reasonable hypothesis might be "predict spam if unknown-sender AND (size OR sales)".

In general, when we are given this kind of question there are two big questions, which are related but distinct. First, how can we automatically generate good hypotheses from the data? It might be a nontrivial computational problem even to find a hypothesis that works on S!. Second, how confident are we that our hypothesis will do well in the future? This is some kind of confidence bound or sample complexity bound – for a given learning algorithm, how much data do we need to see before we can make a guarantee about the future?

Let's begin to formalize this. We are given a sample set $S = \{(x^1, y^1), \ldots (x^m, y^m)\}$, where the examples x^i are drawn from some distribution D and the labels y^i are produced by some target function f. Our algorithm does some kind of optimization over S to produce a hypothesis h, and the goal is to do well on the same distribution D. In other words, we want to create an h so that $\Pr_{x \sim D}[h(x) \neq f(x)] \leq \epsilon$. We call ϵ the *error* of h.

25.2.1 Decision Lists

Let's do an example: learning a decision list. A *decision list* is a function which is essentially one long if/elseif chain, where the first if which is satisfied determines what to return. So, for example, if x is a binary vector then a decision list might be something like "if $x_1 = 1$ then return 0; otherwise if $x_4 = 1$ then return 1; otherwise if $x_2 = 0$ return 1; otherwise return 0". The important features of this are that it doesn't branch (i.e. it's a chain and not a tree, and each if condition looks at just one coordinate.

So suppose that our target function f is a decision list (maybe we have some good reason external to the formalization to suspect that it might be a DL). In order to learn f, we need to do two things. First, we obviously need a way of finding a consistent decision list for our sample S (note that one must exist since we are assuming that f is a DL). But now since we're doing machine learning, we need to also make a second guarantee: we want to say that whatever DL we find for S is actually a good DL for the whole distribution D. In other words, we want to make a guarantee about the future.

We can do this by proving that if $|S|$ is "reasonable", then Pr[exists consistent DL h with $err(h) > \epsilon] < \delta$. Thus the h that we find, since it will be consistent with S, will have error at most ϵ with probability at least $1 - \delta$. This is known as the PAC model, for "probably approximately correct", and is due to Valiant in a seminal paper "A theory of the learnable".

So let's prove that decision lists are PAC-learnable. First, we need an algorithm that generates a DL consistent with S (assuming that one exists). This is reasonably straightforward. We start out with the full list, and find an if-then rule consistent with the data and satisfied by at least one example. This can be done efficiently since there are at most $4n$ possible "if" conditions, so we can just check each one. We then put this condition at the bottom of the list so far, and cross off the examples it covered. We repeat until no examples remain. If this algorithm fails, i.e. if at some point we cannot find a consistent "if" rule, then this clearly means that there is no DL consistent with the remaining data and thus no DL consistent with the original data. Since we are assuming that f is a DL, this can't happen.

So now we have a hypothesis DL. Why do we expect it to do well in the future? Consider some DL h with error more than ϵ that we're worried about. Suppose that S has size m. Then by the definition of errors, the probability that h is consistent with S is at most $(1-\epsilon)^m$. Now let H be the total number of decision lists. Now the union bound implies that Pr[some DL h with $err(h) > \epsilon$ is consistent with $S] \leq H(1-\epsilon)^m$.

We want this value to be at most δ. So if we set m to be at least $\frac{1}{\epsilon}(\ln(H)+\ln(1/\delta))$, then this is satisfied, since then

$$\begin{aligned} H(1-\epsilon)^m &\leq He^{-\epsilon m} \\ &\leq He^{-\ln(H)-\ln(1/\delta)} \\ &\leq \delta \end{aligned}$$

In our case we know that $H \leq n! \cdot 4^n$ since for each feature there are 4 possible rules, and no feature will appear twice in a DL. So the number of samples we need in order to have a DL with error at most ϵ with probability at least $1-\delta$ is only $\Theta(\frac{1}{\epsilon}(n \ln n + \ln(1/\delta)))$.

25.2.2 Occam's Razor

One really nice thing about this analysis is that it didn't actually use anything special about decision lists. All we needed was the number of decision lists to be reasonably small, in this case approximately $n!$. So this analysis would work for *any* class of functions where there aren't too many possible functions to choose from! We just need the number of examples to be essentially $\ln H$, which is a huge improvement over a bound like H. Less formally, we proved that "if there aren't too many different rules to choose from, then it's unlikely that one will fool us just by chance".

This leads to a nice formalization of the classical "Occam's razor". William of Occam, in approximately 1320 AD, made the claim that in general, we should prefer simpler explanations to more complicated ones. While this intuitively makes sense, why should we follow it? What's the benefit of simplicity?

Suppose that "simple" means that the description is at most s bits long. Then there can be at most 2^s simple explanations. Thus our analysis implies that if we see at least $\frac{1}{\epsilon}(s\ln(2)+\ln(1/\delta))$ examples, it is unlikely that one will fool us just by chance. Of course, we don't have any guarantee that there will be a consistent simple explanation, but if there is one then we can be pretty confident in it. Thus we should prefer simple explanations to complicated ones.

25.2.3 Followup work

There have been 1000s of paper using this general framework and making various kinds of improvements. Probably the most interesting direction (at least to me) is replacing the $\ln(H)$ with something like “effective number of degrees of freedom”. For example, consider linear separators. There are an infinite number of linear separators (so H is infinite), but only a small number of really distinct ones given S. This can be formalized, and other more refined analyses can be done.

25.3 Online Learning

What if we don’t want to assume that there is some fixed distribution D? Then clearly we can no longer talk about past performance predicting anything in the future. Is there anything at all that we can say? A statistics-based view would probably say that we’re dead in the water – if the adversary controls the examples we see and the future, then we can’t make any statistically-based guarantees. But from an algorithmic point of view, we can actually make some very interesting claims. The main idea is something called “regret bounds”, and the idea is to show that our algorithm does nearly as well as the best predictor in some large class of predictors.

We’ll illustrate this using the classic setting of “using expert advice”. Suppose that we want to predict whether the stock market will go up or down tomorrow. There are many experts who will tell us what they think will happen, but obviously they might disagree with each other. But we want to use their advice to make a prediction. How can we do this, and what kind of guarantees can we make? The basic question that we’ll consider is whether we can do nearly as well as the best expert in hindsight. Note that this is seems extremely difficult to do – we don’t know anything about these n experts, and the adversary can control what happens at every time point? So one expert might be good for a while and then become terrible, and another expert might start off terrible but get better, etc. Nevertheless, we’ll want to guarantee that we do almost as well as the *best* single expert.

25.3.1 Perfect expert

Let’s start with a simpler setting: suppose that we know one of the experts is perfect (never makes a mistake), we just don’t know which one. Is there a strategy which will let us identify this always-correct expert without making too many mistakes?

Let’s do the following: each day, we take a majority vote of the experts and use that as our prediction. Then we eliminate any experts that were wrong. It is easy to see that this algorithm will only make at most $\log n$ mistakes. This is because every time it makes a mistake, that means that at least half of the remaining experts made the incorrect prediction, so we will eliminate at least half of the remaining experts. Since we started with n experts, we can only eliminate at least half of the remaining experts $\log n$ times before we are left with one expert. And since we are assuming that there is a perfect expert, this one will never be eliminated.

This gives a “mistake bound” of $\log n$.

25.3.2 No perfect expert

If there is no perfect expert then making a mistake doesn't completely disqualify an expert, so we don't want to completely eliminate experts who make a mistake. Instead, we'll give each expert a "weight", and decrease the weight of experts who make mistakes.

This idea gives the *Weighted Majority* algorithm:

1. Initialize all experts to weight 1.

2. Predict based on weighted majority vote.

3. Penalize mistakes by cutting weights in half.

To analyze this, let M be the number of mistakes that we have made so far, let m be the number of mistakes that the best experts has made so far, and let W be the total weight (which starts at n and decreases throughout the algorithm). When we make a mistake, that means that at least half of the current weight gets decreased by half, so W drops by at least 25%. So after we've made M mistakes, W is at most $n(3/4)^m$. On the other hand, the weight of the best expert is exactly $(1/2)^m$. Thus

$$\begin{aligned}(1/2)^m &\leq n(3/4)^M \\ (4/3)^M &\leq n2^m \\ M &\leq \log_{4/3}(n2^m) = (m + \log n)/\log(4/3) \approx 2.4(m + \log n)\end{aligned}$$

It turns out that with improved settings/tweaks, can get $M \leq 1.07m + 8 \ln n$.

This is a pretty good result the number of mistakes that we make is linear (with a reasonably small constant) in the number of mistakes made by the best expert. But what if the best expert is still wrong 20% of the time? Then our algorithm is only doing slightly better than random guessing! (Even if we ignore the $\log n$ part, a mistake bound of $2.4m$ would mean that we make mistakes 48% of the time).

It turns out that we can do a bit better by using randomness. In particular, we will interpret weights as *probabilities*: instead of deterministically taking the weighted majority vote, we will randomly choose an expert according to its weight and then do whatever the expert does. Intuitively, this "smoothes out" the worst case – if the weighted vote is close to balanced, then the adversary won't be able to force us to make a mistake. This is known as the *Randomized Weighted Majority* algorithm. We will also change the penalty for being wrong from 1/2 to ϵ to make the tradeoff between the factor more clear, but we can always go back to setting $\epsilon = 1/2$ if we want.

Theorem 25.3.1 *The expected number of mistakes M made by the randomized weighted majority algorithm is at most $(1 + \epsilon/2)m + \frac{1}{\epsilon} \ln n$.*

Proof: Suppose that at time t there is an F_t fraction of the weight on experts who make a mistake. Then since we always remove an ϵ fraction of the weight from experts who make mistakes, the total weight after time 1 is $F_1 n(1-\epsilon) + (1-F_1)n = n(1-F_1+F_1-\epsilon F_1) = n(1-\epsilon F_1)$. Similarly, the total weight after time 2 is $n(1-\epsilon F_1)(1-\epsilon F_2)$. In general, the total weight after t steps is $n\prod_{i=1}^{t}(1-\epsilon F_t)$. Let W_t be the total weight after t time steps. Then taking logs, we get that

$$\begin{aligned} \ln(W_t) &= \ln n + \sum_{i=1}^{t} \ln(1-\epsilon F_1) \\ &\le \ln n - \epsilon \sum_{i=1}^{t} F_t, \end{aligned}$$

where we used the fact that $\ln(1-x) < -x$. Now note that by linearity of expectations, $\sum_{i=1}^{t} F_t$ is exactly M, the expected number of mistakes that we make! Thus we have that

$$\ln W_t \le \ln n - \epsilon M$$

If the best expert makes m mistakes, then its weight at time t will be $(1-\epsilon)^m$. Thus $\ln W_t \ge m\ln(1-\epsilon)$, and thus $\ln n - \epsilon M \ge m\ln(1-\epsilon)$. Solving for M, we get that

$$M \le \frac{1}{\epsilon}(\ln n - m\ln(1-\epsilon)) \approx (1+\epsilon/2)m + \frac{1}{\epsilon}\ln n$$

■

By setting ϵ to be small, we can achieve a mistake bound that is almost the same as the optimal expert plus something logarithmic in n!

This is a very useful tool to have. For example, we can think of each expert as a different algorithm, and this lets us guarantee that we do almost as well as the best in hindsight. It can even be used for seemingly far-remove uses, such as designing extremely fast approximate max-flow algorithms (due to Garg-Konemann) and fast algorithms for approximately solving LPs. In both of those settings, we essentially have an expert for each constraint and its weight is how much it is violated. Following high-weight experts corresponds to satisfying those constraints.

There are also many extensions to the basic setting. For example, it is easy to extend to essentially arbitrary payoff functions, rather than just yes/no decisions. And with that extension, we can generalize to the so-called "bandit" setting where we only find out the payoff of the expert that we actually choose – the payoffs of the other experts are unknown. We can also handle "sleeping" experts, in which at every time only a subset of the experts actually make a prediction and we need to compare to the best "coalition" of experts (this can be formalized, but we won't do it here). We can also handle things like movement costs, where there is a cost associated with switching the expert that we choose. This is an active area of research with many applications, and there are many, many more extensions. I, for example, have written a few papers that use experts algorithms to maximize the throughput of a wireless network.

25.4 Final notes

There are many other models of machine learning in which we can actually design algorithms with provable guarantees. For example, in "active learning" we are given a huge *unlabeled* sample S and the algorithm can decide which examples to actually label. Similarly, we could allow the algorithm to construct its own examples and ask for them to be labeled (these are called “membership queries”).