15-859(A) Machine Learning Theory

Avrim Blum (avrim@cs.cmu.edu) Lecture 4: January 22, 2004

```
Online learning contd
* The Perceptron algorithm
* The Winnow algorithm
* Recap of results so far
```

A note on material from last time: one implication of our RWM bounds is that for the case when there is a perfect expert, randomized halving (predict using a random consistent expert/hypothesis) gets a bound of $\ln n$ on the expected number of mistakes, compared to the $\lg n$ bound of the standard halving algorithm. Here's an alternate way to see that:

- Let's number the experts by when they make their first mistake, breaking ties arbitrarily. Notice *we only make a mistake if the expert we are using is making its first mistake.*
- The chance we were using expert #1 when it made its first mistake is $1/n$. For expert #2, it's at most $1/(n-1)$. [the probability is $1/n$ if #1 and #2 made their first mistakes at the same time, and is $1/(n-1)$ otherwise]. For expert $\#i$ is $1/(n-i+1)$.
- So, our total expected number of mistakes is at most:

$$1/n + 1/(n-1) + ... + 1/3 + 1/2 = H_n - 1 < \ln n.$$

Plan for today. Last time we looked at algorithms for doing nearly as well as the best "expert". But if we think of experts as features, we might not expect *any* single feature to be a good predictor. Instead, it's more natural to imagine there might be a good *linear combination*, that is, a good linear separator. Today we'll talk about two online linear-separator algorithms.

1 The Perceptron Algorithm

One of the oldest algorithms used in machine learning (from the early 60s) is an online algorithm for learning a linear threshold function called the Perceptron Algorithm.

For simplicity, we'll use a threshold of 0, so we're looking at functions like:

$$w_1x_1 + w_2x_2 + ... + w_nx_n > 0.$$

We can simulate a nonzero threshold with a "dummy" input x_0 that is always 1, so this can be done without loss of generality. The guarantee we'll show for the Perceptron Algorithm is the following:

Theorem 1 *If the data is consistent with a linear threshold function $w^* \cdot x > 0$, where w^* is a unit-length vector, then the number of mistakes is at most $(1/\sigma)^2$, where*

$$\sigma = \min_x |w^* \cdot x|/|x|.$$

(I.e., if we scale examples to have Euclidean length 1, then σ is the minimum distance of any example to the plane $w^* \cdot x = 0$.)

The parameter "σ" is often called the "margin" of w^*. Another way to view the quantity $|w^* \cdot x|/|x|$ is that it is the cosine of the angle between x and w^*, so we will also use $\cos(w^*, x)$ for it.

The Perceptron Algorithm:

1. Start with the all-zeroes weight vector w. Also, let's scale all examples x to have (Euclidean) length 1, since this doesn't affect which side of the plane they are on.
2. Given example x, predict positive iff $w \cdot x > 0$.
3. On a mistake, update as follows:
   - Mistake on positive: $w \leftarrow w + x$.
   - Mistake on negative: $w \leftarrow w - x$.

So, this seems reasonable. E.g., $(w + x) \cdot x = w \cdot x + 1$, and similarly $(w - x) \cdot x = w \cdot x - 1$, so we are moving closer (by 1) to the value we wanted.

Proof of Theorem 1. We're going to look at the magic quantities $w \cdot w^*$ and $|w|$.

Claim 1: every time we make a mistake, $w \cdot w^*$ goes up by at least σ.

> Proof: if x was a positive example, then we get $(w + x) \cdot w^* = w \cdot w^* + x \cdot w^* \geq w \cdot w^* + \sigma$ (by definition of σ). Similarly, if x was a negative example, we get $(w - x) \cdot w^* = w \cdot w^* - x \cdot w^* \geq w \cdot w^* + \sigma$. So, after M mistakes, $w \cdot w^* \geq \sigma M$.

Claim 2: every time we make a mistake, $|w|^2$ goes up by at most 1.

> Proof: if x was a positive example, we get $(w + x)^2 = w^2 + 2(w \cdot x) + x^2$. This is less than $w^2 + 1$ because $w \cdot x$ is negative (remember, we made a mistake on x). Same thing (flipping signs) if x was negative but we predicted positive. So, after M mistakes, $|w| \leq \sqrt{M}$.

Now, we just need to use the fact that $w \cdot w^* \leq |w|$, since w^* is a unit vector. So, if M is the number of mistakes we made, $\sigma M \leq \sqrt{M}$, so $M \leq 1/\sigma^2$. ∎

Discussion: in the worst case, σ can be exponentially small in n. On the other hand, if we're lucky and the data is well-separated, σ might even be large compared to $1/n$. This is called the "large margin" case.

What if only *most* of the data is separable by a large margin, or what if w^* is not perfect? We can see that the thing we need to look at is Claim 1. Claim 1 said that we make "σ amount of progress" on every mistake. Now it's possible there will be mistakes where we make very little progress, or even negative progress. One thing we can do is bound the total number of mistakes we make in terms of the total distance we would have to move the points to make them actually separable by margin σ. Let's call that TD_σ. Then, we get that after M mistakes, $w \cdot w^* \geq \sigma M - \mathrm{TD}_\sigma$. So, combining with Claim 2, we get that $\sqrt{M} \geq \sigma M - \mathrm{TD}_\sigma$. We could solve the quadratic, but this implies, for instance, that $M \leq 1/\sigma^2 + (2/\sigma)\mathrm{TD}_\sigma$.

So, this is not too bad: we can't necessarily say that we're making only a small multiple of the number of mistakes that w^* is (in fact, the problem of finding an approximately-optimal separator is NP-hard), but we can say we're doing well in terms of the "total distance" parameter.

2 The Winnow Algorithm

Now we'll turn to an algorithm that is something like a combination of the WM algorithm and the Perceptron algorithm.

Look at in this context: Say we're trying to learn an OR function. We saw an algorithm: "list all features and cross off bad ones on negative examples" that can make n mistakes. But, what if most features are irrelevant. What if the target is an OR of r relevant features where r is a lot smaller than n. Can we get a better bound in that case?

Winnow will give us a bound of $O(r \log n)$ mistakes. Notice this is proportional to the number of bits needed to *describe* the target function, using $\log n$ bits to name each of the relevant variables.

Winnow Algorithm: (a simple version)

1. Initialize the weights $w_1, \ldots, w_n$ of the variables to 1.
2. Given an example x, output 1 if $w \cdot x \geq n$, and output 0 otherwise.
3. If the algorithm makes a mistake:
   (a) If the algorithm predicts negative on a positive example, then for each x_i equal to 1, double the value of w_i.
   (b) If the algorithm predicts positive on a negative example, then for each x_i equal to 1, cut the value of w_i in half.

Theorem 2 *The Winnow Algorithm learns the class of disjunctions in the Mistake Bound model, making at most $2 + 3r(1 + \lg n)$ mistakes when the target concept is a disjunction of r variables.*

Proof. Let us first bound the number of mistakes that will be made on positive examples. Any mistake made on a positive example must double at least one of the weights in the target function (the *relevant* weights), and a mistake made on a negative example will *not* halve any of these weights, by definition of a disjunction. Furthermore, each of these weights can be doubled at most $1 + \lg n$ times, since only weights that are less than n can ever be doubled. Therefore, Winnow makes at most $r(1 + \lg n)$ mistakes on positive examples.

Now we bound the number of mistakes made on negative examples. The total weight summed over all the variables is initially n. Each mistake made on a positive example increases the total weight by at most n (since before doubling, we must have had $w_1x_1 + \ldots w_nx_n < n$). On the other hand, each mistake made on a negative example decreases the total weight by at least $n/2$ (since before halving, we must have had $w_1x_1 + \ldots + w_nx_n \geq n$). The total weight never drops below zero. Therefore, the number of mistakes made on negative examples is at most twice the number of mistakes made on positive examples, plus 2. That is, $2 + 2r(1 + \lg n)$. Adding this to the bound on the number of mistakes on positive examples yields the theorem. ■

We can also look at the case where data is not completely consistent: if an example satisfies t terms of the target but is labeled negative, this could cause us to cut t relevant weights in half, which could require t mistakes on positives to repair, each one of which causing 2 more mistakes on negatives (for a total of $3t$ extra mistakes).

3 Summary

Here is a list of the online learning results we have so far. (n = number of variables, m = number of mistakes of best function in the class. MB model assumes $m = 0$).

Efficient algorithms:

- Conjunctions/disjunctions: for MB model, list-and-cross-off gives a mistake-bound of n. Winnow gives a mistake-bound of $O(r \log n)$. For general m, Winnow makes $O(rm + r \log n)$ mistakes.
- Single-variable functions (like $f(x) = x_7$): for MB model, halving algorithm gives a mistake-bound of $\lg n$. for the general case, Randomized WM gives an expected mistake bound of $(1 + \epsilon/2)m + (1/\epsilon)(\ln n)$.
- Decision lists: In the MB model, we get MB of $O(nL)$ for list of size L. (hwk)
- LTFs: MB of Perceptron is $1/\sigma^2$. This can be exponentially large in n, or can be small even when n is large.

Information-theoretic (exp'l-time algorithms):

- General concept class C: same as single-variable case, replacing "n" with "$|C|$".