15-850(C) Machine Learning Theory

Lecturer: Avrim Blum
Scribe: Seán Slattery

Lecture 8: October 12, 1995

---

1 Inconsistent Data in the Mistake Bound Model

Up until now we have assumed that our data is completely consistent. What happens to our previous results when we allow for inconsistent data? Generally, for simpler targets we can make better guarantees.

1.1 The Perceptron Algorithm

Here, the target function is defined by a plane (defined by a unit vector w^*) through the origin such that $w^* \cdot x > 0$ for all positive examples x. We saw by reflecting all negative examples through the origin to positive ones, and normalising the lengths of the examples, that if all the examples satisfy $\cos(w^*, x) > \delta$, then the number of mistakes would be $\leq \frac{1}{\delta^2}$.

What happens when we no longer assume that the data is consistent with the correct hypothesis? Say we have M examples, and we plot them on a chart from -1 to 1 according to $\cos(w^*, x)$. Let M_δ be the size of the largest subset of examples S satisfying:

$$\sum_{x \in S} \cos(w^*, x) < \delta$$

We can achieve an error rate of less than M_δ / M in time polynomial in $1/\delta$.

If we organise the algorithm as batch updates (add all x that we make a mistake on to w before re-testing), then we can bound the number of updates required. Remember that the cosine of the angle between w and w^* was given by:

$$\frac{w^* \cdot w}{|w|}$$

Now note that on every update, if our error rate is higher than the above bound, the numerator increases by at least δ, while the square of the denominator increases by at most M. Hence after U updates, $\cos(w^*, w)$ is at least $U\delta/\sqrt{UM}$. We have the correct concept when the cosine is 1. Working this through, we see that we require at most M/δ^2 updates.

1.2 Winnow Algorithim

Recall that Winnow learns a disjunction of r out of n variables.

For an example that satisfies no terms of the target disjunction, but is labeled positive, we will make a mistake on it and add at most n to the total weight. This weight will get

added to terms which are not in the target disjunction and could cause us to misclassify some subsequent negative examples as positive. We saw before that when we misclassify a negative example as positive, the total weight drops by at least $n/2$. Hence it could require up to 2 more mistakes on negatives to repair this damage.

For an example that satisfies t terms of the target disjunction, but is labeled negative, we will make a mistake on it and cut in half t relevant weights. This could require up to t mistakes on positives to repair. And by the analysis of making mistakes on positive examples, it could take 2 more mistakes on negatives for each mistake on a positive to repair. Hence we could make up to an extra $3t$ mistakes after each such erroneous negative example.

1.3 Single Variable Concepts

We now discuss the very simple case of learning concepts defined by a single variable. One example of such learning is a set of financial advisors who make predictions about whether the market will go up or down. We get information from all of them and make our decision about what to do. Eventually we see what the market did do and we know which experts were right. What we would like is to earn as much money as the best expert. So the learning task is to find the best expert (single variable) while not making many mistakes.

The Winnow analysis tells us that if the best expert made m mistakes, then Winnow would make at most $4m + O(\log n)$ mistakes. However, when dealing with only single variable concepts, we can do a better analysis with a simpler algorithm called the *weighted majority* algorithm [1].

Weighted Majority - Version 1

The algorithm proceeds as follows:

- Start with each expert having weight 1.
- Given an example, poll all the experts and predict the outcome with the highest total weight.
- After every prediction, cut in half the weights for all the experts who made the wrong prediction.

To analyse this, let M be the total number of mistakes we've made so far, W be the total weight and m be the number of mistakes the best expert so far has made. Now, when we make a mistake, we know that more than $W/2$ of the weight was on the wrong choice. Hence, after a mistake, the total weight decreases by at least $W/4$. Therefore, after M mistakes, $W \leq n(\frac{3}{4})^M$. The weight of the best expert is just $(\frac{1}{2})^m$. Now we observe that the total weight cannot be less than the weight of the best expert (since the weights can never be below zero), so we get:

$$n\left(\frac{3}{4}\right)^M \geq \left(\frac{1}{2}\right)^m$$

$$\begin{aligned}\left(\frac{4}{3}\right)^M &\leq n2^m \\ M &\leq 2.4(\log_2 n + m)\end{aligned}$$

Weighted Majority – Version 2

This version treats the weights as probabilities. Now instead of doing majority vote using the weights, we predict outcome i with probability w_i/W, where W is the total weight and w_i is the total weight associated with outcome i. A practical advantage of this approach is that we can equivalently select one expert with probability proportional to its weight and return its prediction. This speeds up prediction, although we also need to look at each expert's prediction in order to make an update.

We will analyse a more general form of the algorithm where the weight of each wrong expert is multiplied by $\beta < 1$ instead of $1/2$. Define F_i to be the fraction of the total weight on the *wrong* answers at the i^{th} example. Say we've seen t examples. Let M be our expected number of mistakes so far, so $M = \sum_{i=1}^{t} F_i$.

Now, on the i^{th} example, the total weight changes as:

$$W \leftarrow W(1 - (1-\beta)F_i)$$

since we multply the weights of experts that got it wrong by β and there is an F_i fraction of the weight on these experts.

Hence the final weight is:

$$W = n(1-(1-\beta)F_1)(1-(1-\beta)F_2)\cdots(1-(1-\beta)F_t)$$

Let's say that m is the number of mistakes of the best expert so far, and again use the fact that the total weight must be at least as large as the weight on the best expert, so $W \geq \beta^m$

So we have:

$$n\prod_{i=1}^{t}(1-(1-\beta)F_i) \geq \beta^m$$

Now we take the natural log of both sides and solve for M which we said previously was $\sum F_i$:

$$\begin{aligned}\ln n + \sum_{i=1}^{t}\ln(1-(1-\beta)F_i) &\geq m\ln\beta \\ -\ln n - \sum_{i=1}^{t}\ln(1-(1-\beta)F_i) &\leq m\ln(1/\beta) \\ -\ln n + (1-\beta)\sum_{i=1}^{t}F_i &\leq m\ln(1/\beta) \\ M &\leq \frac{m\ln(1/\beta)+\ln n}{1-\beta}\end{aligned}$$

Where we get the third line by noting that $\ln(1-x) > x$. So for $\beta = 1/2$, we get an expected number of mistakes of less than $1.39m + 2\ln n$, and for $\beta = 3/4$ we get an expected number of mistakes of less than $1.15m + 4\ln n$.

2 Generalising Winnow

Suppose instead of a disjunction of r variables, we have a majority-type function. The target now specifies r variables and some integer t. An example is positive if at least t of the r variables are on, and is negative otherwise. Hence the function looks like:

$$w_1 x_1 + w_2 x_2 + \cdots + w_n x_n \geq t$$

where r of the weights are 1 and the rest are 0.

Let's say that we know t. Now we set $\alpha = 1 + 1/(2t)$ and multiply the weights by α when we make a mistake on a positive example, and divide by α when we make a mistake on a negative. Think of multiplying by α as putting a poker chip on the weight, and dividing as removing the chip (negative chips are allowed).

The maximum number of chips on the r relevant variables is $r \log_\alpha n$, since if the weight ever gets to n, then it can never increase as we can't make any more mistakes on positives.

Now a mistake on a positive example causes us to put at least t chips on relevant weights, and the total weight increases by at most $n(\alpha - 1)$, since it is possible that all variables were on and hence all the weights got multiplied by α.

A mistake on a negative example takes at most $t - 1$ chips off relevant weights. In this case the total weight comes down by at least $n(1 - 1/\alpha)$.

Using these facts we can construct the following two inequalities in two unknowns:

$$\begin{aligned} t(M \text{ on positive}) - (t-1)(M \text{ on negative}) &\leq r \log_\alpha n \\ n + (M \text{ on positive}) n(\alpha - 1) &\geq (M \text{ on negative}) n(1 - 1/\alpha) \end{aligned}$$

Where the second inequality just states that the total weight should be at least 0.

These two are pretty messy to solve, but end up giving the total number of mistakes to be $O(tr \log n)$.

Of course, we may not know t to begin with. However, if we know r, then we can make a guess at t and double it. If we don't know r, then we need to make a guess at both of them and double it, and then the cost becomes $O(r^2 \log n)$.

References

[1] N. Littlestone and M. K. Warmuth. The weighted majority algorithm. *Information and Computation*, 108(2):212–261, 1994.