CS787: Advanced Algorithms

Scribe: Dan Wong, Blayne Field **Lecturer:** Shuchi Chawla
Topic: MB Model, Perceptron Alg. and PAC Model **Date:** November 14, 2007

24.1 Introduction

Continuing from the last lecture, we resume our discussion of concept learning in the mistake bound (MB) model. Recalling from last time, we want to give an online algorithm for learning a function from a concept (aka hypothesis) class C. We are given examples with n features, and we wish to predict the value of our target function. After each step, we learn if our prediction was correct or not, and update our future predictions accordingly. Our goal is to do this in a manner that guarantees some bound on the number of mistakes that our algorithm makes.

Definition 24.1.1 *A class C is said to be learnable in the MB model if there exists an online algorithm that given examples labeled by a target function $f \in C$ makes $O(log|C|)$ prediction mistakes and the run time for each step is bounded by $O(n, log|C|)$*

It turns out that this requirement for the run time is necessary, since without it, the halving algorithm (discussed in a previous lecture) would give us a $O(log|C|)$ bound for any concept class C. It was demonstrated in the last lecture that the class of boolean OR functions is learnable in the MB model, along with boolean OR functions with at most k literals. We also saw that the winnow algorithm (described below) solves (r of k) majority functions in the MB model.

24.2 Winnow Algorithm

Consider the problem of learning a (r of k) concept class. Here the class we are trying to learn is one where the target function contains k attributes and if at least r of them are present in the example then it is true otherwise it is false.

Recall the Winnow Algorithm:

- Start with $w_{i,0} = 1 \forall i$
- Predict 1 if $\sum w_{i,t} x_{i,t} \geq n$ else predict 0.
- Update weights on mistake as follows:
– Positive Example: For every i with $x_{i,t} = 1$ set $w_{i,t+1} = (1+\epsilon) w_{i,t}$
– Negative Example: For every i with $x_{i,t} = 1$ set $w_{i,t+1} = \frac{w_{i,t}}{(1+\epsilon)}$

Note that if the weight of an attribute is greater than n, it will never increase further.

Let P = The number of mistakes on positive examples, and likewise N be the number of mistakes on negative examples.
Recall: $\sum_i w_{i,0} = n$

The total increase in $\sum_i w_i \leq P\epsilon n$
The total decrease in $\sum_i w_i \geq \frac{N\epsilon n}{1+\epsilon}$

Since our weights never go negative we get an inequality:

$$n + P\epsilon n \geq \frac{N\epsilon n}{1+\epsilon}$$

To make another inequality we will count the total number of increases or decreases to any relevant attribute seen by our set of k attributes. Consider the analogy of throwing coins in buckets again where every bucket is one of the relevant attributes and every time we increase an attributes weight we add a coin and every time we decrease its weight we remove one. We add at least r new coins for each positive example since there must have been at least r target attributes in our example since it was labeled positive. For each negative example we remove at most $(r-1)$ coins since there must have been at most $(r-1)$ target attributes since the example was labeled negative. The maximum net increase seen by any single one of the k attributes is $\log_{1+\epsilon} n$.

Therefore the number of increases minus the decreases should be less than or equal to the maximum net increase:

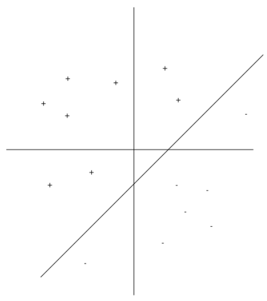
$$Pr - (r-1)N \leq klog_{(1+\epsilon)}n$$

Solving the 2 inequalities for P and N yields:

$$P, N = O(rk \log n), \text{for } \epsilon = \frac{1}{2r}$$

24.3 Linear Threshold Functions

One class of functions that we are interested in learning is that of linear threshold functions. A linear threshold function in n dimensions is a hyperplane that splits the space in two, where the points on different sides correspond to positive and negative labels. The function is defined by a set of weights $W = \{w_1, w_2, ..., w_n\}$, where the label of the function then becomes $sign[\sum w_i x_i - w_0]$. This type of function may arise in many applications, such as if we were given examples of people



by their height and weight, labeled by whether they were obese, and we wished to learn a function of height and weight that predicted whether a person is obese.

Note that for this class of functions, we can't apply the definition of being learnable, since the class has infinite size. Instead, we will derive a mistake bound based on the geometric properties of the concept class.

24.4 Perceptron Algorithm

Suppose we are given some target function f from the class of linear threshold functions (which we will assume WLOG that $w_0 = 0$). We want to have an algorithm to learn this function with a reasonable bound on the number of errors. It turns out that such an algorithm exists (the perceptron algorithm), which gives us a bound on the number of errors as a function of the margin of the target function. If we define $\hat{w_f}$ to be the normal of the plane defining our target function, then $\gamma(f) = min_{x:|x|=1}|x \cdot \hat{W_f}|$ is known as the margin of f. At a high level, the perceptron algorithm works by maintaining a vector w (denoted at time t by w_t). At each step, we make a prediction according to $sign(w \cdot x)$, where x is our input.

The Perceptron Algorithm goes as follows:
- Start with $w_0 = \vec{0}$
- At each step, predict the label of x_t according to $sign(w_t \cdot x_t)$.
- If we made a mistake, set $w_{t+1} = w_t + l(x_t)x_t$, where l_t is the true label of x_t
- Otherwise, set $w_{t+1} = w_t$ (make no change)

It turns out that the perceptron algorithm makes no more than $\frac{1}{\gamma^2}$ mistakes, which we prove below.

Lemma 24.4.1 *If we make a mistake at step t, then* $w_{t+1} \cdot \hat{w_f} \geq w_t \cdot \hat{w_f} + \gamma$

Proof: To proof Lemma 24.4.1, we just go to the definition:
$w_{t+1} \cdot \hat{w_f} = (w_t + l(x_t)x + t) \cdot \hat{w_f} = w_t \cdot \hat{w_f} + l(x_t)x_t \cdot \hat{w_f} \geq w_t \cdot \hat{w_f} + \gamma$. Since $|x \cdot \hat{w_f}| \geq \gamma$ by definition. Also, we know that signs work out correctly, since if we made a mistake, $l(x_t)$ has the same sign as $x_t \cdot \hat{w_f}$ ■

Alone, this lemma does not neccesarily mean we are approaching the correct value, since the dot product could just be getting larger due to the size of the vector increasing. In the next lemma, we show that the lengths of the vectors don't grow very rapidly.

Lemma 24.4.2 *If step t is a mistake, then* $||w_{t+1}||^2 \leq ||w_t||^2 + 1$.

Proof: To proof Lemma 24.4.2, once again we just use the definition: $||w_{t+1}||^2 = ||w_t + l(x_t)x_t||^2 = ||w_t||^2 + ||x_t||^2 + 2w_t \cdot x_t l(x_t)$, since $w_t x_t$ and $l(x_t)$ have opposite signs, so $2w_t \cdot x_t l(x_t) < 0$. ■

Combining these two lemmas gives us that after M mistakes $\sqrt{M} \geq |w_t| + 1 \geq w_t \cdot \hat{w_f} \geq \gamma * M \rightarrow M \leq \frac{1}{\gamma^2}$

24.5 Extensions to the prediction problem

It also turns out that we extend the weighted majority algorithm to solve variations of the prediction problem as well. These variations include the situation where we don't learn the costs of the experts we did not pick (a very real possibility in a real-world situation). It may also be the case that not all experts are available or relevant at all time steps, and the algorithm must be modified to deal with these changes. Another possible situation is that we may want to assign some random initial offset to our experts.

24.6 Introduction to PAC Model

The PAC Model, Probably Approximately Correct Model, is a learning model where we claim that the function we are trying to learn is approximately correct.

Definition 24.6.1 *A concept class, C, is PAC learnable if there exists an algoirthm, A, such that $\forall \epsilon, \delta > 0$ for every distribution D, and any target function, $f \in C$, given a set S of examples drawn from D labled by f, A produces with probability $(1-\delta)$ a function h such that $\mathbf{Pr}_{x \sim D}\left[h(x) = f(x)\right] \geq (1-\epsilon)$.*

Note that by allowing the constructed function h to differ from f with some probability ϵ, we are protected against negligible weight examples. In a similar sense δ protects against the event that when sampling S from D that we get an unrepresentative set S. Since low weight examples are unlikely to be seen in the sample, it would be unreasonable to expect the algorithm to perform well in either case.

In addition we require the algorithm to work in reasonable amount of time and only use a reasonable amount of examples.
Specifically:

The size of S,$|S|$ should be $poly(\frac{1}{\epsilon}, \frac{1}{\gamma}, n, \log|C|)$
Runtime should be $poly(\frac{1}{\epsilon}, \frac{1}{\gamma}, n, \log|C|)$