CS 395T Computational Learning Theory

Lecture 8: October 5, 2005 Lecturer: Adam Klivans

Scribe: Shivaram Kalyanakrishnan

8.1 Learning Halfspaces: Background

Over the last few lectures, we have studied different algorithms for learning halfspaces. These include linear programming, perceptron, and Winnow. It turns out that while these are all reasonably good algorithms, they have some limitations as well.

Using linear programming to solve for the coefficients of the threshold function becomes prohibitively expensive for a large training set. The perceptron algorithm is quite simple, but is not guaranteed to converge in polynomial time. It especially suffers when noisy examples are present in the data, and also when the margin of separation is small. The Winnow algorithm only works well when the

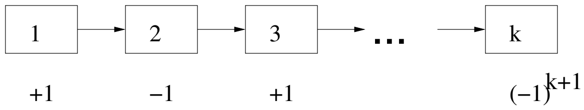

Figure 8.1: An Alternating Decision List.

total weight of the threshold function being learned is bounded polynomially. (Note that we assume that the coefficients of the threshold function corresponding to a halfspace are integers; this is a reasonable assumption because we usually only have finite precision to represent them.). In general, the coefficients need not be polynomial in size; for instance, an 'alternating' decision list with k nodes (depicted in Fig. 8.1) is represented by the halfspace $h = SIGN(2^{k-1}x_1 - 2^{k-2}x_2 + 2^{k-3}x_3 + \cdots + (-1)^{k+1}x_k)$, whose coefficients are seen to be exponential. In fact, a result due to Hastad[2] establishes that there exist halfspaces for which the magnitude of the coefficients of must be $\Omega(n^n)$. Surely then, Winnow will have very poor performance while learning such halfspaces.

In today's lecture, we study a very simple algorithm for learning halfspaces. Called the *Average* algorithm, it makes certain assumptions about the distribution of the data. Under these assumptions, it performs extremely well, and is shown to be PAC.

In his Ph.D. thesis, Rocco Servedio[3] provides a wonderful description of the *Average* algorithm. The presentation in the following section is largely based on Servedio's description, but ignores issues like the presence of noise that find place in his analysis.

8.2 The *Average* Algorithm

In this section we specify and analyse the *Average* algorithm.

8.2.1 Problem Definition

The *Average* algorithm makes two simplifying assumptions to the general problem of learning a halfspace:

1. That the points are distributed uniformly on the unit hypersphere S^{n-1}, and
2. That the separating halfspace is origin centred.

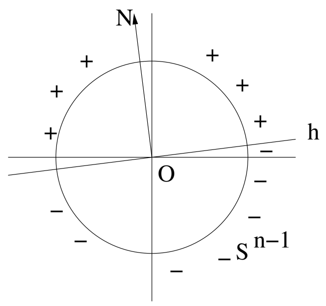

Figure 8.2: The Problem.

These assumptions are depicted in Fig. 8.2. For the sake of imagining a uniform distribution on S^{n-1}, we mention that if one were to pick each dimension of a point in n dimensional space based on a gaussian with mean 0 and variance $\frac{1}{n}$, the point *approximately* corresponds to one drawn uniformly from the surface of S^{n-1}. We also emphasise here that by constraining our training data to come from a uniform distribution on a hypershpere, we are solving a less general problem than we would if the points were allowed to be drawn from an arbitrary distribution.

Our second assumption, that the halfspace is origin centred, is not a limiting assumption. This is so because if some halfspace separates the set of data in n dimensions, then certainly we can add a dummy dimension to all the data points and have an origin centred halfspace in a dimension one higher separate them.

8.2.2 Input and Output

Posed with the problem of finding a halfspace that separates the set of positive points from the set of negative points, we make the following simplification. We remove from our training set every negative example x_{neg}, and in its place add to our training data set a positive example $-x_{neg}$; i.e., we reflect each negative example about the origin and label the reflection a positive example. This is illustrated in Fig. 8.3. It is apparent that any halfspace that separates the positives from the negatives in our original data set will contain all the positives (both original and reflected negatives) on one side, and conversely, any halfspace that has all the positives on its one side after reflection will separate the positives from the negatives in the original data set. Therefore, since we can use reflection to transform any data set into an equivalent one containing only positives, we have essentially established that it suffices to learn an origin centred halfspace that just contains all the (positive) training examples on its one side. Henceforward, we will assume that our input set only comprises positive examples.

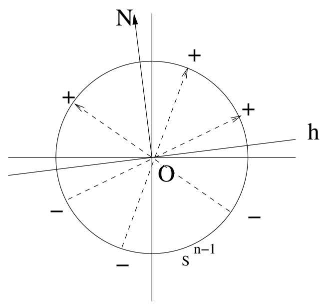

Figure 8.3: Reflection of Negative Examples about the Origin.

Since our output is an origin centred halfspace, it is completely represented by its normal vector.

8.2.3 *Average*

The *Average* algorithm is described as follows:

- Given the input points $X_1, X_2, X_3, ..., X_t$, present as output their average: $\frac{1}{t}\sum_{i=1}^{t} X_i$.

We proceed to analyse the performance of this simple operation.

8.2.4 Analysis

We will make use of three facts during our analysis. These are listed below.

Fact 1 *If x is drawn uniformly from S^{n-1}, then for any fixed vector u on S^{n-1}, and real numbers α, β $(-1 \leq \alpha \leq \beta \leq 1)$,*

$$Pr\,(\alpha \leq u \cdot x \leq \beta) \quad = \quad \frac{A_{n-2}}{A_{n-1}} \int_{\alpha}^{\beta} \left(\sqrt{1-z^2}\right)^{n-3} dz,$$

where A_m represents surface area of S^m. Using the formula $A_m = \frac{\pi^{\frac{m}{2}}}{\Gamma\left(\frac{m}{2}\right)}$, and using the Stirling approximation for $\Gamma\left(\frac{n}{2}\right)$, we get

$$Pr\,(\alpha \leq u \cdot x \leq \beta) \quad \approx \quad \sqrt{n} \int_{\alpha}^{\beta} \left(\sqrt{1-z^2}\right)^{n-3} dz.$$

Fact 2 *We may assume, without loss of generality, that our unknown halfspace u corresponds to a unit vector along the first dimension, i.e., $u = (1, 0, 0, 0, ..., 0)$. This assumption does not restrict generality because the argument that follows is independent of the specification of the coordinate axes. We merely use it to simplify our arguments.*

Fact 3 *Let the correct hypothesis be U, and let it correspond to the vector u. Let V be some hypothesis, such that its corresponding vector v makes an angle θ with u. The error of the hypothesis V is given by*

$$\begin{aligned} Error\,[V] &= Pr_{x \epsilon S^{n-1}}\left(V(x) \neq U(x)\right) \\ &= Pr_{x \epsilon S^{n-1}}\left(SIGN(v.x) \neq SIGN(u.x)\right) \\ &= \frac{\theta}{\pi}. \end{aligned}$$

This follows from geometry.

In our analysis, we use the notation we introduced in Fact 3. Let u be the vector corresponding to the correct hypothesis U (recall that this is $(1, 0, 0, 0, ..., 0)$). Let V be out current hypothesis, and corresponding to it v, the vector our averaging algorithm has computed after seeing n examples. And let θ be the angle between u and v. As our first step, we compute the vector v', which corresponds

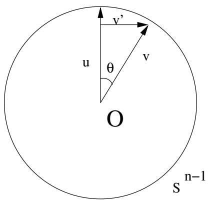

Figure 8.4: Error of a Hypothesis.

to v's component orthogonal to u, i.e., $v' = v - (u \cdot v)\, u$. v' is shown in Fig. 8.4. Using Fact 3, our error is given by

$$Error\,[V] \;=\; \frac{1}{\pi} tan^{-1}\left(\frac{\|v'\|}{u \cdot v}\right). \tag{8.1}$$

The strategy we employ to prove that this error is small is to show that the *parallel component* $u \cdot v$ is large, and that the *orthogonal component* v' is small.

A Large Parallel Component

Assume that we take t examples, drawn uniformly from S^{n-1}. Let x be one such example, and let its label be b (b is 1 or -1). Because we initially reflect negative examples about the origin, the input vector corresponding to the example x is bx.

Now, using Fact 1,

$$E\,[u \cdot bx] \;\approx\; 2\sqrt{n} \int_0^1 \left(\sqrt{1-z^2}\right)^{n-3} dz.$$

The factor 2 appears because the negative points get reflected into the same halfspace as the positive points. We then argue that

$$2\sqrt{n} \int_0^1 \left(\sqrt{1-z^2}\right)^{n-3} dz \;\geq\; 2\sqrt{n} \int_0^{\frac{1}{\sqrt{n}}} \left(\sqrt{1-z^2}\right)^{n-3} dz$$

$$\begin{aligned} &\geq 2\sqrt{n}\int_0^{\frac{1}{\sqrt{n}}} czdz, \ \textit{for some large constant } c \\ &= 2\sqrt{n}c\frac{1}{n}, \ \textit{for some large constant } c \\ &= \Omega(\frac{1}{\sqrt{n}}). \end{aligned}$$

It follows, therefore, that

$$E\left[u \cdot bx\right] = \Omega(\frac{1}{\sqrt{n}}).$$

Since v is the average of t such independently chosen x's, it follows, by linearity of expectation, that

$$E\left[u \cdot v\right] = \Omega(\frac{1}{\sqrt{n}}).$$

Hoeffding's tail bound applies to the average X of t independently chosen random variables lying in $[-1, 1]$:

$$Pr\left[x \leq \frac{1}{2}E\left[X\right]\right] \leq 2e^{-\frac{tE[X]^2}{8}}.$$

Since $u \cdot v$ is precisely such an average, it follows that we can bound the above probability by δ if we take $t = \Theta\left(nlog\frac{1}{\delta}\right)$ examples. That is, choosing $t = \Theta\left(nlog\frac{1}{\delta}\right)$ examples guarantees that with probability δ, $u \cdot v = \Omega(\frac{1}{\sqrt{n}})$.

$$\textit{With high probability}, \ u \cdot v = \Omega(\frac{1}{\sqrt{n}}). \tag{8.2}$$

A Small Orthogonal Component

For any example x, the component orthogonal to u, i.e., $x - (u \cdot x)\, u$, will have the form $(0, c_2, c_3, c_4, \ldots, c_n)$, since $u = (1, 0, 0, 0, \ldots, 0)$. Since x is drawn uniformly from S^{n-1}, it follows that with high probability, $c_i, for\ i = 2, 3, 4, \ldots, n$, will have magnitude $O(\frac{1}{\sqrt{n}})$. This is derived by applying the argument used in Fact 1; the components $c_2, c_3, c_4, \ldots, c_n$ are nothing but dot products of x with unit vectors along the respective dimensions.

Now, we can establish, using a Hoeffding tail bound, that if one were to walk a distance a to the right on heads and a to the left on tails of an unbiased coin, then with high probability, one would be within $\sqrt{t}a$ of the origin after t tosses. Applying this to our case, we find that the the sum tv' of the orthogonal components of t random examples should have its first dimension 0, and every other dimension, with high probability, $\sqrt{t}\frac{1}{\sqrt{n}}$. Therefore, its magnitude must be $O(\sqrt{t})$.

$$\textit{With high probability}, \ \|tv'\| = O(\sqrt{t}). \tag{8.3}$$

Final Result

Having established that v has a large parallel component$(u \cdot v)$ and a small orthogonal component(v'), we combine these to bound our error (from Eqn. 8.1):

$$\begin{aligned} Error\,[V] &= \frac{1}{\pi} tan^{-1}\left(\frac{\|v'\|}{u \cdot v}\right) \\ &\approx \frac{1}{\pi}\left(\frac{\|v'\|}{u \cdot v}\right),\ (\left(\frac{\|v'\|}{u \cdot v}\right)\ is\ small) \\ &= O(\frac{\sqrt{n}}{\sqrt{t}}),\ (using\ Eqn.s\ 8.2\ and\ 8.3). \end{aligned}$$

Therefore, to bound with high probablity the error by ϵ, we need only take $t = O(\frac{n}{\epsilon^2})$ examples. The simple *Average* algorithm is shown to have good performance.

8.3 Learning Intersections of Halfspaces

Fig. 8.5 pictorially represents the intersection of two halfspaces. Learning intersections of halfspaces under arbitrary distributions remains an open problem, but Baum[1] has proposed a method to do so when the distribution is symmetric, i.e., $D(x) = D(-x)$. A sub-part of the solution involves

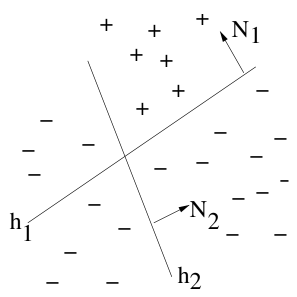

Figure 8.5: Intersection of Two Halfspaces.

learning the XOR of two halfspaces.

8.3.1 Learning the XOR of Two Halfspaces

Fig. 8.6 depicts the XOR of two halfspaces. Let's assume we have to learn the XOR of the two halfspaces f and g, represented by the linear threshold functions $L_f(x) = \sum_{i=1}^{n} \alpha_i^f x_i - \theta^f$, and $L_g(x) = \sum_{i=1}^{n} \alpha_i^g x_i - \theta^g$ respectively. That is, $f(x) \equiv SIGN(L_f(x))$, and $g(x) \equiv SIGN(L_g(x))$. Consequently,

$$\begin{aligned} (f \oplus g)(x) &\equiv SIGN(L_f(x)) \oplus SIGN(L_g(x)) \\ &= -SIGN(L_f(x) \cdot l_g(x)) \end{aligned}$$

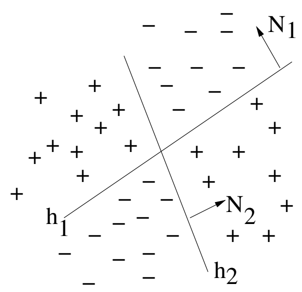

Figure 8.6: XOR of Two Halfspaces.

$$= \quad -SIGN((\sum_{i=1}^{n} \alpha_i^f x_i - \theta^f) \cdot (\sum_{i=1}^{n} \alpha_i^g x_i - \theta^g)).$$

It is easy to show that if we introduce new variables $y_k$to correspond to the products $x_i x_j, i \neq j$, and also cover the n x_i's, then the right hand side can be expressed as a linear function over these ${}^nC_2 + n$ variables, i.e.,

$$\begin{aligned} (f \oplus g)(x) &\equiv SIGN(L_h(x)), where \\ L_h(x) &= \sum_{i=1}^{{}^nC_2+n} \alpha_i^h y_i(x) - \theta^h. \end{aligned}$$

Thus, to learn the XOR of two halfspaces in n dimensions, we may simply transform the problem into $O(n^2)$ dimensional space as specified above, and learn a halfspace in the new space. Our algorithm remains polynomial.

8.3.2 Extending XOR to Intersection

In the next lecture we discuss how we can easily extend the solution to learning the XOR of two halfspaces to learning their intersection, under the assumption that the distribution of the data is symmetric.

References

[1] E. B. Baum. On learning a union of half spaces. *J. Complex.*, 6(1):67–101, 1990.

[2] Johan Hastad. On the size of weights for threshold gates. *SIAM J. Discret. Math.*, 7(3):484–492, 1994.

[3] Rocco Servedio. *Efficient Algorithms in Computational Learning Theory.* PhD thesis, 2001.