CS 2416 - Machine Learning Theory

Lecture #3: 29 September 2005

Lecturer: Toniann Pitassi

Scribe Notes by: Toniann Pitassi

1 The Perceptron Algorithm

We will continue from last time by describing one more algorithm for learning in the MB model. This algorithm, called the perceptron algorithm, is one of the oldest machine learning algorithms (from the early 60's). It is an online/MB algorithm for learning a linear threshold function. (A linear threshold function is a function of the form $w_1x_1 + w_2x_2 + \ldots + w_nx_n > w_0$, where the w_i's are rational numbers.

For simplicity we will assume that the threshold (w_0) is always 0, so the functions in our concept class look like: $w_1x_1 + w_2x_2 + \ldots + w_nx_n > 0$. (We can simulate a nonzero threshold by introducing a new dummy variable x_0 which is always equal to 1.)

To visualize what is going on, let $W =< W_1, W_2, \ldots, W_n >$ be a point in the n-dimensional space, and assume that the length of W, $|W|$, is 1. Without loss of generality, we will assume that all examples x have Euclidean length 1. Then since $Wx = |W||x| \cos\theta = \cos\theta$, we want to accept a particular x if the angle between W and x is within +/- 90. Let P be the $n-1$ dimensional hyperplane that is perpendicular to the line passing through $\overline{0}$ and W. Then the cosine of the angle between W and x will be within 90 degrees exactly if x lies on the w-side of the hyperplane P. In 2 dimensions, draw a picture.

The Perceptron algorithm is as follows. Start with the all zero weight vector w. Predict positive if and only if $wx > 0$. (Both w and x are vectors so this is a dot product.) If we make a mistake on a positive example, then set the new weight as follows $w = w + x$. If we make a mistake on a negative example, then set $w = w - x$.

Notes. At each stage, the algorithm predicts according to its current hypothesis threshold function defined by w. Notice that although we will assume in the analysis that the target threshold function (defined by W) has length 1, the hypothesis threshold function (defined iteratively by w) will not necessarily have length 1. Note that whenever we make a mistake, we are moving closer to the desired value by 1, since $(w + x)x = wx + 1$ and similarly $(w - x)x = wx - 1$.

Theorem 1 *If the data is consistent with a linear threshold function $Wx > 0$ where $|W| = 1$, then the number of mistakes made by the perceptron algorithm is at most $(1/\delta)^2$ where $\delta = min_x|Wx|$.*

Proof The proof is obtained from the following two facts.

Fact Every time we make a mistake, wW goes up by at least δ.

To see this, suppose that x was a positive example. Then we have $(w+x)W = wW + xW \geq wW + \delta$, by the definition of δ. Similarly if x was a negative example then $(w-x)W = wW - xW \geq wW + \delta$. So after M mistakes, $wW \geq \delta M$.

Fact Every time we make a mistake, $|w|^2$ goes up by at most 1. (Note that $|w| = \sqrt{w_1^2 + \ldots + w_n^2}$.)

Again if x was a positive example then we have $(w+x)^2 = w^2 + 2(wx) + x^2$ which is less than $w^2 + 1$ since wx is negative. A similar argument holds when x is a negative example. Thus after M mistakes, $|w| \leq \sqrt{M}$.

To conclude the proof of the theorem, notice that $wW \leq |w|$ since $wW = |w||W|cos\theta = |w|cos\theta \leq |w|$, where the second equality holds because W is a unit vector. (This is the only place where we need W to be a unit vector.) Thus after M mistakes we have:

$$\delta M \leq wW \leq |w| \leq \sqrt{M}.$$

Solving for M yields

$$M \leq (1/\delta)^2.$$

2 PAC learning model

Definition Let $\mathcal{C}$ be a concept class (concepts over n variables), and let $\mathcal{D}$ be a distribution over the input space. Algorithm A is a randomized algorithm that has access to an oracle which returns labelled examples from some unknown $c \in \mathcal{C}$ where the examples are taken from distribution $\mathcal{D}$. Algorithm A PAC-learns concept class $\mathcal{C}$ by hypothesis class $\mathcal{H}$ if for any $c \in \mathcal{C}$, for any distribution $\mathcal{D}$ over the input (instance) space, any $\epsilon, \delta > 0$, A runs in time $poly(n, 1/\epsilon, 1/\delta, size(c))$ (and thus also uses at most poly many samples) to produce a hypothesis $h \in \mathcal{H}$ such that: with probability at least $(1-\delta)$, the error between h and c over $\mathcal{D}$ is at most ϵ. This probability is taken over the random coin tosses of A as well as over the random labelled examples seen from distribution $\mathcal{D}$.

The quantity ϵ is called the accuracy parameter and δ is called the confidence parameter. A hypothesis with error at most ϵ is sometimes called ϵ-good.

When $\mathcal{H} = \mathcal{C}$ (the hypothesis must be some concept in $\mathcal{C}$) then this is called proper PAC learning.

2.1 Example: Learning monotone conjunctions

The algorithm that we presented before to learn monotone conjunctions in the consistency model is also a valid PAC algorithm. The PAC algorithm A is as follows. Given ϵ, δ and n, choose m examples drawn from distribution $\mathcal{D}$. (The value of m will be chosen later, but it will be polynomial in $1/\delta \cdot 1/epsilon \cdot n$ as required.) Run the consistency-model algorithm on these samples and output the associated hypothesis. We now want to argue that A PAC learns the class of monotone conjunctions with error ϵ and confidence δ. Fix a particular concept c from the concept class $\mathcal{C}_n$, and fix the distribution $\mathcal{D}_n$. Let h be a particular bad hypothesis with respect to c and $\mathcal{D}_n$. That is, assume that $error_{\mathcal{D}}(h, c) \geq \epsilon$. We want to compute the probability that A outputs h (when the examples are randomly chosen from $\mathcal{D}_n$ according to c).

The probability that h is output by A is equal to the probability that on all m samples, h and c agree. This probability is at most $(1-\epsilon)^m$. Thus the overall probability that A outputs a bad conjunction (when the examples are randomly chosen from $\mathcal{D}_n$ according to c) is at most $2^n(1-\epsilon)^m$. Setting this to be at most δ, and using the inequality $(1-\epsilon)^{1/\epsilon} \leq 1/e$, we have: $2^n(1/e)^{\epsilon m} \leq \delta$ or equivalently, $e^{\epsilon m} \geq 2^n/\delta$. Taking logs gives $m \geq (1/\epsilon)[nln(2) + ln(1/\delta)]$. As long as m is larger than this quantity, our algorithm finds a consistent conjunction with probability at least $1-\delta$, as desired.

2.2 Relationship between PAC and Consistency/Occam model

Theorem 2 *Let A be an algorithm that learns concept class $\mathcal{C} = \cup_{\backslash}\mathcal{C}_{\backslash}$ by hypothesis class $\mathcal{H} = \cup_{\backslash}\mathcal{H}_{\backslash}$ in the consistency model. Then there is a PAC algorithm A' using only $(1/\epsilon)(ln|\mathcal{H}_n| + ln(1/\delta))$ examples to learn $\mathcal{C}_n$ (but A' may not be efficient).*

The proof of the above theorem is nearly the same as the proof that the algorithm for learning monotone conjunctions is a PAC learning algorithm. Namely, the probability that a bad hypothesis is consistent with m samples is at most $(1-\epsilon)^m$. So the probability that there exists a bad consistent hypothesis (by the union bound) is at most $|\mathcal{H}_n|(1-\epsilon)^m$. Setting this equal to δ and solving, this says that the number of samples is at most $(1/\epsilon)[ln|\mathcal{H}_n| + ln(1/\delta)]$.

Occam learning is learning in the consistency model, but with the extra condition that we want as succinct a representation as possible. In particular, we say that a concept class $\mathcal{C}$ has an OCCAM learning algorithm if there exists an algorithm A such that for any set of samples, A produces a hypothesis h from some class $\mathcal{H}$ that is consistent with the samples, and furthermore the size of h is sublinear in the size of the sample set.

Definition Let $\alpha \geq 0$ and $0 \leq \beta < 1$. L is an (α, β)-Occam algorithm for $\mathcal{C}$ using $\mathcal{H}$ if for all n, on input a sample set S of cardinality m labelled according to $c \in \mathcal{C}_n$, L outputs a hypothesis $h \in \mathcal{H}_n$ such that: (1) h is consistent with S; and (2) $size(h) \leq (n \cdot size(c))^\alpha m^\beta$. L is an *efficient* (α, β)-Occam algorithm if its running time is bounded by a polynomial in n, m and $size(c)$.

Note that when a concept class has an Occam algorithm, then compression is occurring. Remember from the first class (the case of learning DNF), that if our goal is only to output some hypothesis that is consistent with the samples, then we can simply memorize the samples. But Occam learning must output a hypothesis that is substantially smaller than the size of the samples, and thus the labelled samples are being compressed by h.

Theorem 3 *(Occam's Razor) Let L be an efficient (α, β)-Occam algorithm for $\mathcal{C}$ using $\mathcal{H}$. Let $\mathcal{D}$ be the target distribution over the instance space $X = x_1, \ldots x_n$ and let $c \in \mathcal{C}_n$ be the target concept, $0 < \epsilon, \delta \leq 1$. Then there is a constant $a > 0$ such that if L is given as input a random sample S of m examples drawn according to $\mathcal{D}$ and c, where m satisfies*

$$m \geq a \left(\frac{1}{\epsilon} \log \frac{1}{\delta} + \left(\frac{(n \cdot size(c))^\alpha}{\epsilon} \right)^{\frac{1}{1-\beta}} \right)$$

then with probability at least $1-\delta$ the output h of L has error at most ϵ. Moreover, L runs in time polynomial in n, $size(c)$, $1/\epsilon$ and $1/\delta$.

Proof Let $\mathcal{H}_{n,m}$ be the set of possible hypothesis representations that the (α, β)-Occam algorithm L may output when given as input a labelled sample S, $|S| = m$. Since L is an (α, β)-Occam algorithm, each hypothesis has bit length at most $(n \cdot size(c))^\alpha m^\beta$, and therefore $|\mathcal{H}_{n,m}| \leq 2^{(n \cdot size(c))^\alpha m^\beta}$.

For a fixed bad hypothesis h (one where $error(h) > \epsilon$), the probability that h is consistent with m examples from $\mathcal{D}$ is at most $(1-\epsilon)^m$. Using the union bound, this implies that the probability that some bad hypothesis is consistent with a random sample from $\mathcal{D}$ of size m is at most $|\mathcal{H}_{n,m}|(1-\epsilon)^m$. We want to this to be at most δ.

This gives:

$$m \log(\frac{1}{1-\epsilon}) \geq \log(|\mathcal{H}_{n,m}|) - \log(\frac{1}{\delta})$$

Solving for m (where $\log(\frac{1}{1-\epsilon}) = b\epsilon$), we get

$$m \geq \frac{1}{b\epsilon} \log |\mathcal{H}_{n,m}| + \frac{1}{b\epsilon} \log \frac{1}{\delta}.$$

Since the right hand side is no greater than $2max(k, k')$, where $k = \frac{1}{b\epsilon} \log |\mathcal{H}_{n,m}|$ and $k' = \frac{1}{b\epsilon} \log \frac{1}{\delta}$, as long as $m \geq 2max(k, k')$, the above inequality is satisfied. Choosing $a = 2/b$ yields the statement of the theorem.

Here is another version of the theorem given in the book.

Theorem 4 *(Occam's Razor, Cardinality version) Let $\mathcal{C}$ be a concept class and $\mathcal{H}$ a representation class. Let L be an algorithm such that for any $c \in \mathcal{C}_n$, if L is given as input a sample S of m labelled examples of c, then L runs in time polynomial in $n, m, size(c)$ and outputs an h in $\mathcal{H}_{n,m}$ that is consistent with S. Then there is a constant $b > 0$ such that for any n, any distribution $\mathcal{D}_n$ over $\{0,1\}^n$, any $c \in \mathcal{C}_n$, if L is given as input m samples chosen according to $\mathcal{D}_n$ where $m \geq (1/b\epsilon)(\log |\mathcal{H}_{n,m}| + \log(1/\delta))$, then L will find some $h \in \mathcal{H}_{n,m}$ such that with probability at least $1-\delta$, $error(h) \leq \epsilon$.*