CS 2416 - Machine Learning Theory

Lecture #4: 6 October 2005

Lecturer: Toniann Pitassi

Scribe Notes by: Toniann Pitassi

In the last class, we defined the PAC model, and showed its connection to the consistency/Occam models. Today we will describe how to get a PAC algorithm from a MB algorithm, and then give several examples of concept classes that are PAC-learnable.

1 Relationship between PAC and Mistake-bound models

Theorem 1 *Let A be an algorithm for learning C in the MB model with mistake bound M. Then we can PAC learn C using a training set of size $O(M/\epsilon * log(M/\delta)$.*

Let $E = O(M/\epsilon * log(M/\delta)$. To prove this theorem, we will run our MB algorithm on the data set of size E, and look at the sequence of at most M hypotheses produced, $h_1, h_2, \ldots, h_M$. We can assume without loss of generality that a new hypothesis is produced by the MB algorithm only when it makes a mistake on an example. Therefore, on average there will be $s = (1/\epsilon)log(M/\delta)$ examples in between any two different hypotheses. We will choose the first hypothesis, h_i, such that the number of examples between h_i and h_{i+1} is at least this number. We want to argue that this hypothesis, h_i, PAC learns C over the same distribution.

How can our algorithm go wrong? First, it goes wrong if h_1 is chosen (so its empirical error on the next s samples is zero), but its actual error is greater than ϵ. Since these s samples are chosen at random, the probability that h_1 (with actual error at least ϵ) gets all of the examples correct between h_1 and h_2 is at most $(1-\epsilon)^{(1/\epsilon)log(M/\delta)}$ which is δ/M. A second way in which our algorithm can go wrong is if h_1 is not chosen and h_2 is chosen. The bad probability here is the probability that h_2 is chosen, times the probability that h_1 is not chosen given that h_2 is chosen. This is at most the probability that h_2 is chosen. Again by the above calculation, this happens if the actual error of h_2 is at least ϵ but h_2 gets all of the random s examples correct between h_2 and h_3. Again, this probability is at most δ/M. Continuing on, the total bad probability is upper bounded by the sum over each i, of the probability that h_i has actual error ϵ but gets s random samples correct (where the probability is over the s random samples). This is at most $M(\delta/M) = \delta$.

The above bound on the training set size can be improved to $O((1/\epsilon)[M + \log(1/\delta)])$.

2 PAC-learnable Concepts

2.1 Learning conjunctions with few relevant variables

Here we'll give another algorithm for learning conjunctions (or disjunctions) with few relevant variables. In the algorithm that we discussed before, our representation of each conjunction was a

$2n$ bit vector, with value 1 corresponding to those literals that occur in the conjunction. However, we may want a different representation where we explicitly list each literal in the conjunction. In this more succinct representation, a conjunction of size k over n variables will be represented by a bit vector of length $k\log(2n)$. With this new representation, we now desire an OCCAM algorithm that finds a hypothesis that is close in size to the target concept class. We present such an algorithm here.

Recall the original algorithm in the consistency model for learning conjunctions. We begin with the full conjunction $h = x_1 \neg x_1 x_2 \neg x_2 \ldots x_n \neg x_n$. We then go through all of the positive examples, throwing out a literal from h only if it is set to 0 in (at least) one of the positive examples. The negative examples were never used to prune h. We will now describe how to further prune h by using the negative examples. Let S denote a set of m negative examples. For each negative example, we know that at least one of the variables that is set to 0 must be in the conjunction. We can think of our problem as an instance of the set cover problem, as follows. The underlying universe, U, is m elements, one corresponding to each negative example. For each literal l occurring in h, let N_l denote the set of negative examples where literal l is set to 0. The subsets, S, are N_l, for each $l \in h$. If l is in the conjunction h, then it "covers" all negative examples from N_l. We thus want to find a minimal set of subsets (literals), $T \subseteq S$, such that $\{N_l \mid l \in T\}$ covers all of U.

Recall the following facts about the set cover problem. (Given as input a collection S of subsets of $U = \{1, \ldots m\}$, find a subcollection $T \subseteq S$ such that $|T|$ is minimuzed and the sets in T form a cover of U. That is, the union of the sets in T equals U.) While this problem is NP-hard, the natural greedy algorithm runs in polynomial-time, and produces a cover R of cardinality at most $O(opt(S)\log m)$, where $opt(S)$ is the cardinality of the minimal cover. (The greedy algorithm first chooses the largest set, s^*, from S as the first element in the cover. After choosing this set, S is updated by replacing each set s in S by $s - s^*$. The algorithm repeats until all elements from U are covered by R.)

Applying the greedy algorithm for set cover described above to our problem, we obtain a cover of size $size(c)\log m$. Thus our hypothesis class is the set of all conjunctions of at most $size(c)\log m$ literals, and thus each hypothesis in $H_{n,m}$ can be described with $size(c)\log m \log n$ many bits. We thus have an (α, β)-Occam algorithm for some $\alpha \geq 0$ and $0 \leq \beta < 1$, and therefore by the above theorem, the concept class is also PAC-learnable.

Note that the fewer the number of negative examples, the better off we are. In the above, we assumed the worst case situation where there are m negative examples.

2.2 Learning decision trees

A decision tree over $x_1, \ldots, x_n$ is a binary tree. Each internal node in the tree is labelled with a variable x_i, and the two edges out of the node labelled x_i are labelled by $x_i = 0$ and $x_i = 1$ respectively. The leaves of the decision tree are labelled either 0 or 1. A decision tree over $x_1, \ldots, x_n$ computes a boolean function f_n in the following way. For any truth assignment α, there is exactly one path in T that is consistent with α; the value at the leaf of α is the value of the function represented by T on input α. The size of a decision tree T is simply the number of nodes in T.

Theorem 2 *The class of decision trees is PAC learnable in time $n^{O(\log s)}$ where s is the size of the target concept/decision tree.*

Proof We describe an Occam-style algorithm for learning. Let T be a particular decision tree over n variables $x_1, \ldots, x_n$ of size s. We begin with a set S of m labelled examples of T and we want to find a decision tree T' consistent with all examples from S. The algorithm performs the following in parallel for each of the $2n$ literals (or in a dovetailing fashion): Creates a partial decision tree with the root labelled the variable x underlying the literal, and the edge labelled by $x = 0$ if the literal is $\neg x$, and $x = 1$ if the literal is x. This partitions the set of labelled examples S into two disjoint sets: those examples consistent with $x_i = 0$ (S_0), and those examples consistent with $x_i = 1$ (S_1).

If all examples in S_0 are labelled in the same way, then we can terminate this branch of the partial decision tree and label it accordingly. Otherwise, there is at least one example in S_0 labelled by 1, and at least one example in S_0 labelled by 0, so we must continue recursively. In this case, the new variable set is X minus x, and the set of examples is S_0. Similarly, we check to see if all examples in S_1 are labelled in the same way, and if so we can terminate this branch, and otherwise we must continue recursively.

Let $T(n, s)$ be the runtime required to find a tree consistent with a set of samples over n variables, when we know that there exists such a tree of size at most s.

Then we have $T(1, s) = 1$, $T(n, 1) = 1$, and $T(n, s) \leq 2n[T(n-1, s/2)] + T(n-1, s)$. To justify the last inequality, let T^* be the optimal tree of size at most s that is consistent with the samples. The root of T^* is labelled with some variable x_j, and since T^* has size at most s, one of the two subtrees below the root has size at most $s/2$.

$$\begin{aligned} T(n, s) &\leq 2n[T(n-1, s/2) + T(n-2, s/2)] + T(n-2, s) \\ &\leq 2n[nT(n-1, s/2)] \\ &\leq n^{O(logs)} \end{aligned}$$

Now we can apply the theorem proven last class to go from this OCCAM algorithm to a PAC algorithm. This algorithm is a simplification/modification of the usual algorithm by Rivest for learning decision lists, which implies a similar algorithlm for learning decision trees. Note that the above algorithm is not an efficient PAC algorithm since its runtime is quasipolynomial instead of polynomial. However, it is still relatively efficient. Note further that the algorithm is actually a proper PAC algorithm since the hypothesis class is also the class of all decision trees.