CS 2416 - Machine Learning Theory

Lecture #5: 13 October 2005

Lecturer: Toniann Pitassi

Scribe Notes by: Toniann Pitassi

1 Learning DNF

In today's lecture, we will give the best known proper PAC algorithm for learning DNF formulas. A DNF over $x_1, \ldots, x_n$ is a disjunction of terms, where a term is a conjunction of literals. We will represent a term by a bit vector of length $2n$. Our representation of a DNF formula of n variables will be the concatenation of the bit vectors representing each of the terms in the disjuntion. Thus if the DNF has k terms, its representation will have bit length $k\log(2n)$.

Theorem 1 *The class of DNF formulas can be PAC learned in time $n^{\sqrt{nlogs}}$, where s is the number of terms in the target DNF.*

Proof Again we will describe an OCCAM algorithm for the class of DNF formulas. Let $f = t_1 \vee t_2 \vee \cdots \vee t_s$. Let $w = \sqrt{2n\log s}$. A term (or conjunction) t is said to be wide if $|t| \geq w$. Let E be a set of labelled examples of f. The learning algorithm is as follows. We begin with (E, s, n), where: (1) E is a set of labelled examples consistent with some DNF formula over n variables; (2) we are looking for a DNF f that is consistent with E and with the property that f contains at most s wide terms.

If $s = 0$, then this is our base case, and we will find a w-disjunction consistent with the current set of examples, if one exists, in time $n^{(O(w)}$. Recall from the first lecture, that this can be done in the consistency model. If we don't find such a consisent w-disjunction, then halt this process.

Otherwise, $s \geq 0$, in parallel for each of the $2n$ literals, we want to recurse on the two subproblems (i) $(E\lceil_{l=1}, s(1-\frac{w}{2n}), n-1)$ and (ii) $(E\lceil_{l=0}, s, n-1)$.

The runtime analysis is as follows. Suppose that E is consistent with a DNF, f, of size at most s. Then in particular we know that E is consistent with a DNF containing at most $s' \leq s$ wide terms. If we set a literal to true in f, then this literal occurs in $s'w/2n$ terms on average. Thus, there exists a literal l such that f restricted by $l = 1$ has at most $s(1-\frac{w}{2n})$ wide terms. The recursion is therefore:

$$\begin{aligned} T(n,0) &= n^{O(w)} = n^{O(\sqrt{n\log s})} \\ T(n,s) &\leq 2n[T(n-1, s(1-\frac{w}{2n})] + T(n-1,s) \\ &\leq 2n[nT(n-1, s(1-\frac{w}{2n}))] \end{aligned}$$

$$
\begin{aligned}
&\leq 2n^2 T(n-1, s(1 - \frac{\sqrt{\log s}}{\sqrt{2n}}) \\
&\leq (2n^2)^{\sqrt{2n \log s}} T(n-1, 0) \\
&= n^{O(\sqrt{n \log s})}
\end{aligned}
$$

Note that the above algorithm PAC learns in the proper model. That is, we can view the hypothesis class as also consisting of DNF formulas. (Do you see why?)

Klivans and Servedio get an improved result, but not in the proper model. That is, they show how to PAC learn DNF in time $2^{O(n^{1/3} \log n)}$. Their hypothesis class is larger than the class of all DNF formulas.

Note that if we want to learn size s DNF formulas over the uniform distribution, this can be done in time $n^{(\log s)}$, so this is much easier than over an arbitrary distribution.

Jackson (extending work of KM) show how to learn (in polynomial time) over the uniform distribution, but now the model is an extension of the PAC model where you are additionally allowed membership queries. This is a very nice algorithm that we will discuss later.

1.1 Connection to automatizability algorithms

The algorithm presented above to learn decision trees and DNF formulas are modifications of well-known algorithms in proof complexity, used to show that DPLL and Resolution (respectively) are automatizable. DPLL (Davis-Putnam-Logam-Loveland), is a popular variant of Resolution used in first order as well as in propositional theorem proving. For propositional theorem proving, one begins with an unsatisfiable CNF formula f, and a DPLL-based procedure will create a decision tree that falsifies f. The automatizability result states that there is a deterministic algorithm that is given as input an unsatisfiable formula f, and that outputs a DPLL refutation of f. Furthermore, the runtime of the algorithm is at most $|f|^{O(logs)}$, where s is the size of the optimal DPLL refutation for f.

In both cases (learning decision trees and automatizability of DPLL), the algorithm is trying to search through all possible decision trees. Since the existence of a tree of size s implies that at least one of the subtrees of the tree has size at most $s/2$, this leads to the recursion equation and thus the quasipolynomial runtime, in both cases.

Similarly, the above algorithm to learn DNF in subexponential time is a modification of a similar algorithm used in proof complexity to show that Resolution can be automatized in time $2^{\sqrt{n \log s}}$. As far as we know, this latter result is new. Results in the learning theory literature show how to learn DNF in subexponential time, but not in the proper model.

VC Dimension

A couple of lectures ago, we showed that if a learning algorithm draws a random sample of

$$m = O(\frac{1}{\delta \varepsilon}(\log |H| + \log \frac{1}{\delta}))$$

examples, and finds a hypothesis in C consistent with these examples, then h will also PAC learn the concept class. We obtained the above bound via the union bound. That is, we computed the probability that a fixed bad hypothesis is consistent with the samples, and then took the sum over all possible bad hypotheses. Unfortunately, if the size of the concept class is infinite, then the above argument is not useful. Nonetheless, there are infinite concept classes that can be learned with only polynomially many samples.

VC dimension (Vapnik-Chervonenkis) is a general measure of complexity for concept classes that is particularly useful for concept classes of infinite cardinality. Let C be a concept class over instance space X, where typically both C and X are infinite.

Definition Let S be a set of points/examples. Define $C[S]$ as the number of ways to split S using a concept from the concept class $\mathcal{C}$. Define $C[m]$ as the maximum over all sets S, $|S| = m$, of $C[S]$. That is, $C[m]$ is the maximum number of ways to split m points (examples) using a concept from the concept class $\mathcal{C}$. (The textbook calls this $\pi_C(m)$.) Clearly $C[m] \leq 2^m$.

Definition A set of points S is *shattered* by $\mathcal{C}$ if there are concepts in $\mathcal{C}$ that split S in all of the $2^{|S|}$ possible ways. (That is to say that if you can split every possible combination of input with some concepts from C, then you have shattered S.)

Definition The VC Dimension of $\mathcal{C}$ is the size of the **largest** set of points that can be shattered by $\mathcal{C}$. This is denoted by $VCdim(\mathcal{C})$. $VCdim(\mathcal{C}) = d$ means that there exists a set S, $|S| = d$, that is shattered by $\mathcal{C}$. It also means that there is no set of size $d + 1$ can be shattered.

Examples of VC Dimension

Example - Linear halfspaces in 2D. It's clear that given any 2 non-colinear points, that you can shatter them by a plane, simply by draw a line between, or on either side, of the points, depending on whether they are positive or negative). Given any **3** non-colinear points, they can again be shattered by drawing lines between the points, no matter what the labelling of the points. (You should prove this to yourself if you are skeptical. There are 8 possibilities you should consider: when all examples are positive (negative), when only one of the points is positive (negative), and when two of the points are positive (negative). In each case you will be able to draw a line such that all the positive examples are on one side and all the negative examples are on the other.) However, you can prove given any 4 points, they cannot be shattered in 2D. If these four are arranged in a convex hull formation (picture 4 points in a diamond shape, with positive examples at the top and bottom of the diamond, and negative examples at the left and right), there is no way to divide these points so that all positive examples are on one side of the plane and all negative examples are on another. An equivalent example has three points arranged in a regular triangle, with positive examples at every point, but a fourth negative example at the centre of the triangle. Therefore, $VCdim(\text{linear halfspaces in 2D}) = 3$.

Example - Intervals on the real line. Any two points can be contained:

- if both points are positive examples, contain them.
- if both points are negative examples, contain the area outside of them.
- if one is positive and one is negative, contain only the positive point.

However, any three points cannot be contained. A example of this is a line with a positive example, then a negative example, then a positive example. There is no interval you can create on the real line such that the positive examples will be contained apart from the negative examples.

Example All boolean functions over $x_1, ..., x_n$. Here there are 2^n points, each with 2^n possible values. Because there are 2^n possible inputs, and there's 2^{n^n} boolean functions over n variables, each one different, this means that the VC dimension here is 2^n.