

Lecture 1 : January 20, 2005

Lecturer: Rocco Servedio

Scribe: Rocco Servedio

1 Introduction

Computational learning theory (or more generally, machine learning) has the following goals:

- Develop automated ways to construct computer programs from data.
- Extract useful information efficiently from raw data.

Many learning problems can be classified as one of the following two types:

Supervised learning: Each example consists of a pair (x, y) where x denotes the instance and y its label. The goal is to infer the process being used to label the data. As a motivating example, consider the problem of trying to develop a system for automatically labeling images of handwritten digits. The instance x would be the image file, and the corresponding label $y \in \{0, 1, \dots, 9\}$ would be the correct value of the digit that was written. One would like to develop a program which, after being trained on a sample of labeled instances (i.e. (x, y) pairs), does a good job of correctly predicting the label y' when given a new instance x' .

Unsupervised learning: Each data point is unlabeled; the goal is to come up with a good description of the process that is generating the data. Clustering problems often fall into this framework. As a motivating example, consider the predicament of a botanist on a new island, who would like to come up with a useful taxonomy of the plants and animals that he sees.

This course focuses on supervised learning.

Learning has been studied from many different points of view:

Psychology: How do human beings learn?

Statistics: Much work in statistics deals with building build statistical models for data which can be used for prediction. Currently there is a lot of active research in statistical learning theory on providing detailed analyses of how much information can be extracted from a finite sample (generalization error bounds).

Applied/Machine Learning: This approach could be characterized by the question “How does this algorithm work on my data?” Sophisticated algorithms (SVMs, neural networks, and more) are often used.

Computational Learning Theory/Theoretical Computer Science: This is the point of view we’ll adopt in this course. It focuses on well-defined, abstract learning scenarios, often involving learning *Boolean functions* (i.e. functions $f : \{0, 1\}^n \rightarrow \{0, 1\}$). The function f to be learned is typically assumed to have some “nice structure” such as such as having a succinct representation as a decision tree, a DNF formula, or a Boolean circuit. The goal to explore the capabilities and limitations of different learning algorithms in this abstract context; there is an emphasis on provable correctness, and results are presented in a theorem-proof style.

The focus of this course will be exploring algorithms for “hard” (or rather, “interesting”) learning problems; we shall be very interested in the asymptotic efficiency of our algorithms. The course blends ideas and techniques from the analysis of algorithms and complexity theory.

2 Concept Classes

Reading: “Classes of Boolean Functions” by N. Bshouty (check class webpage for links).

The high-level idea of all the learning models we’ll consider is that a learning problem is defined by a *concept class* C . Such a class C is a set of Boolean functions $f : \{0, 1\}^n \rightarrow \{0, 1\}$; intuitively, C is the set of possible concepts that the learning algorithm may be asked to learn.

There is a “target concept” c_\star that belongs to C ; this is the function that labels the examples used for learning, i.e. the labeled examples the learner receives are of the form $(x, c_\star(x))$. The identity of the target concept is unknown to the learner (though the learner does know that the target concept is some element of C); the goal of the

learner is to identify (exactly or approximately) c_* in a computationally efficient way. (The details of exactly what's required of the learner, as well as the details of how the learner accesses these labeled examples, will vary as we consider different learning models throughout the semester.) The efficiency of a learning algorithm is measured relative to the two parameters n and $\mathbf{size}(c_*)$, where $\mathbf{size}(c)$ is defined differently for different concept classes.

We now describe some particular concept classes that we'll study this semester.

2.1 DNF Formulas

A *DNF formula* (short for **D**isjunctive **N**ormal **F**orm) is an OR of ANDs of Boolean literals. Recall that a Boolean literal is either a Boolean input variable x_i or the negation of a Boolean input variable \bar{x}_i . (Here x_i denotes the i -th bit position of an n -bit input string $x \in \{0, 1\}^n$).

Example: $(x_1 \wedge \bar{x}_3 \wedge x_4 \wedge x_5) \vee (x_2 \wedge x_5 \wedge x_6) \vee (\bar{x}_4 \wedge \bar{x}_7)$ is a DNF formula. Here we write “ \vee ” to denote OR and “ \wedge ” to denote AND. For succinctness we'll often omit the “ \wedge ” when there is no potential for confusion, i.e. we will typically write the above DNF as $(x_1\bar{x}_3x_4x_5) \vee (x_2x_5x_6) \vee (\bar{x}_4\bar{x}_7)$.

The *size* of a DNF is measured by the number of terms (conjunctions) it contains; so the above example is a DNF of size 3, or a 3-term DNF.

From a combinatorial viewpoint, one can consider the elements of $\{0, 1\}^n$ as the vertices of an n dimensional cube. Then each term of a DNF could be identified with the subcube of $\{0, 1\}^n$ consisting of those $x \in \{0, 1\}^n$ which satisfy the term. For instance the term $x_1\bar{x}_3x_4x_5$ in the above DNF could be identified with the subcube $1 * 011 * * \dots *$; here each “ $*$ ” denotes a “wildcard” character that can take either value 0 or 1. It follows that the positive instances of an s -term DNF corresponds to a union of s subcubes of $\{0, 1\}^n$.

DNFs are a *universal representation scheme*, in the sense that any $f : \{0, 1\}^n \rightarrow \{0, 1\}$ can be represented by a DNF.

It is easy to see why this fact holds: for any given $z \in \{0, 1\}^n$, there is a unique term (which is a conjunction of exactly n literals) that is satisfied by z and by no other input. (For instance, if $n = 5$ and $z = 11100$, the term would be $x_1x_2x_3\bar{x}_4\bar{x}_5$.) By taking a disjunction of these terms over all z such that $f(z) = 1$, we obtain a DNF formula which computes f . Of course, there will usually be many other ways

to construct a DNF formula that represents f , perhaps using many fewer terms.

Given any $f : \{0, 1\}^n \rightarrow \{0, 1\}$, the *DNF-size* of f is the number of terms in the smallest DNF (fewest number of terms) that computes f . From the above paragraph it is clear that any function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ has DNF-size at most 2^n . With a little thought this bound can be improved to 2^{n-1} ; note that this 2^{n-1} bound is tight because the smallest DNF computing the parity function over n bits has exactly 2^{n-1} many terms. (Recall that the parity function over n bits outputs 1 if an odd number of the bits are set to 1, and outputs 0 if an even number of the bits are set to 1. In other words, the parity function just computes $x_1 + \dots + x_n \pmod 2$.)

To see that any DNF for the parity function must have 2^{n-1} terms, observe that each term in such a DNF must be of length exactly n (i.e. it must be satisfied by exactly one input string). There are 2^{n-1} input strings that satisfy the parity function, so 2^{n-1} terms are required.

A *k-DNF* is a DNF in which each term contains at most k literals. Note that consequently any k -DNF has at most $\sum_{i=1}^k 2^i \binom{n}{i} \approx n^k$ terms.

2.2 Decision trees

A *decision tree* (often abbreviated DT) is a rooted binary tree in which each internal node is labeled with one of x_1, \dots, x_n and has exactly two children, and each leaf is labeled with a bit (either 0 or 1). The left child of a variable x_i always corresponds to the setting $x_i = 0$ and the right child to $x_i = 1$. A decision tree determines a Boolean function f in the obvious way: given an input $x \in \{0, 1\}^n$, traverse the tree in the obvious way (if you are at a node labeled with x_i , go to its left or right child according to whether x_i equals 0 or 1 as described above) until a leaf is reached; the bit at that leaf is the value of $f(x)$. For example, if we are evaluating the decision tree in figure 1 on input $x = 00110$, we first go the left child of x_1 because $x_1 = 0$, then we go to the right child of x_3 because $x_3 = 1$, and finally output 0 because $x_5 = 0$.

The *size* of a DT is measured by the number of leaves in the DT (note that this is one more than the number of internal nodes). It is easy to see that DTs are a universal representation scheme: any $f : \{0, 1\}^n \rightarrow \{0, 1\}$ can be represented by some DT. (Consider the full DT with 2^n leaves, where all nodes at level i are labeled with variable x_i .)

The *DT-size* of $f : \{0, 1\}^n \rightarrow \{0, 1\}$ is the size of the smallest DT representing f . Clearly the DT-size of any n -variable Boolean function f is at most 2^n . It's not hard to see that the parity function on n variables has DT-size exactly 2^n .

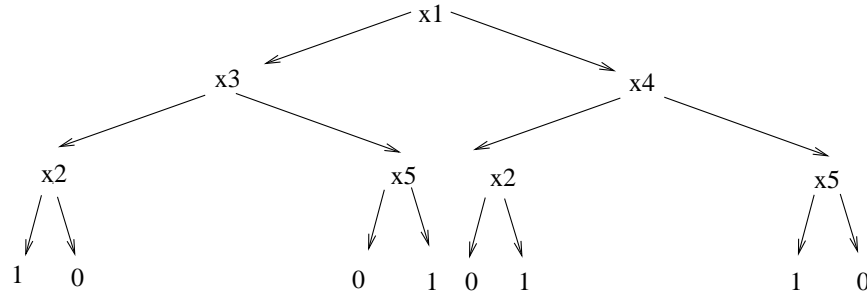


Figure 1: A decision tree

Note that in any given DT, each root-to-leaf corresponds to a conjunction of Boolean literals, and hence to a subcube of $\{0, 1\}^n$. Since every input in $\{0, 1\}^n$ reaches exactly one leaf, it follows that an s -leaf DT gives a partition of $\{0, 1\}^n$ into s disjoint subcubes. (We can view each subcube as being “colored” with either 0 or 1 depending on what the output label is of the associated leaf.)

Lemma 1 *If $c : \{0, 1\}^n \rightarrow \{0, 1\}$ has a size s DT, then*

- *c has an s -term DNF.*
- *c has an s -clause CNF.*

Proof: One can construct a DNF from a DT for c as follows: for each path to a leaf with output bit 1 in the DT, there is a corresponding conjunction of literals as described above. By taking a DNF whose terms are precisely these conjunctions, we obtain a DNF that computes exactly the same function as the DT; such a DNF clearly has at most s terms. (For example, the DNF corresponding to the DT of Figure 1 would be $\bar{x}_1\bar{x}_3\bar{x}_2 \vee \bar{x}_1x_3x_5 \vee x_1\bar{x}_4x_2 \vee x_1x_4\bar{x}_5$). This gives the first result.

For the second result, observe that by flipping all the leaf bits in a DT for c , we obtain a DT for \bar{c} , the negation of c . Using this observation, one can construct an s -term DNF for \bar{c} as described above, and negate it using DeMorgan’s Law to obtain an s clause CNF for c . ■

Note that there do exist functions with small DNF-size but large DT-size. One example of such a function is the DNF

$$x_1x_2 \vee x_3x_4 \vee \cdots \vee x_{n-1}x_n \quad (\text{for } n \text{ even}).$$

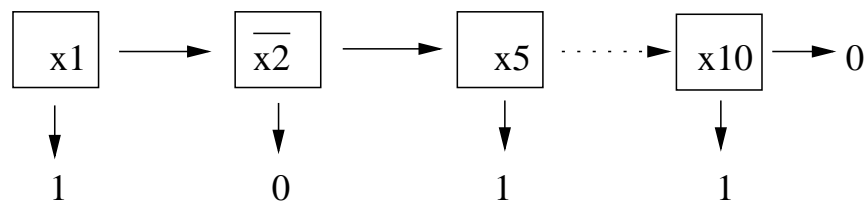


Figure 2: A decision list

One can show that any DT for this function must have at least $2^{\Omega(n)}$ many leaves.

The *depth* of a DT is measured by the length of the longest root-to-leaf path. Any depth- d DT has size at most 2^d and also has a d -DNF (since one can express the function as a disjunction of terms with length d using the paths in the DT as described above).

A *decision list* is a DT with only one nontrivial branch, see Figure 2. A k -*decision list* is a decision list in which each internal node contains a conjunction of up to k Boolean literals.

2.3 Linear threshold functions

In this section we will view Boolean functions as mapping to the range $\{-1, 1\}$ rather than $\{0, 1\}$; this is more convenient for linear threshold functions.

Recall that the sign function $\text{sign}(z)$ takes value 1 if $z \geq 0$ and takes value -1 if $z < 0$.

Definition 1 A function $c : \{0, 1\}^n \rightarrow \{-1, 1\}$ is a linear threshold function (or LTF) if there exist $w_1, \dots, w_n, \theta \in \mathbb{R}$ such that for all $x \in \{0, 1\}^n$, $c(x) = \text{sign}(w_1x_1 + \dots + w_nx_n - \theta)$. We say that such a w, θ are a representation of the LTF.

Example: The function $c(x) = \text{sign}(3x_1 - 4x_2 + 2x_4 - 1)$ is an LTF.

Geometrically, one can view a linear threshold function as corresponding to an n -dimensional hyperplane in \mathbb{R}^n ; the positive instances for such a function are precisely the points of the Boolean cube that lie on one side of this separating hyperplane. With this intuition it is easy to see that LTFs are *not* a universal representation scheme – note that the parity function on two bits, which maps $(0, 0)$ and $(1, 1)$ to 1 and $(1, 0)$ and $(0, 1)$ to -1 is not linearly separable in this sense.

Here are some basic facts about LTFs:

Fact 1 *Any LTF $c : \{0, 1\}^n \rightarrow \{-1, 1\}$ has infinitely many representations.*

This follows from the fact that one can perform a (sufficiently small) perturbation of w or θ without changing which side of the separating hyperplane any point in $\{0, 1\}^n$ lies on. In fact, one can perturb the coefficients for the LTF in such a way that each w_i is a rational number, and then multiply all these coefficients with the common denominator to obtain the following fact:

Fact 2 *Any LTF has a representation with integer coefficients.*

Definition 2 *Given a LTF c , the weight of c is measured by the minimum value of $\sum_i |w_i|$ over all possible integer representations.*

Example: $c = x_1 \vee \dots \vee x_k$ is an LTF since it could be expressed as $\text{sign}(x_1 + \dots + x_k - 1)$. The weight of c is easily seen to be k .

How many distinct LTFs are there over $\{0, 1\}^n$?

Fact 3 *There are at least*

$$(1 + 2) \cdot (1 + 4) \cdot (1 + 8) \cdots (1 + 2^{n-1}) = 2^{\Omega(n^2)}$$

many distinct LTFs over $\{0, 1\}^n$.

Proof sketch: The argument is by considering what happens as we add one dimension at a time to the Boolean cube. If we stipulate that our LTF must pass between the two inputs $(0, 0)$ and $(0, 1)$, then there are three ways that the LTF can label the two points $(1, 0)$ and $(1, 1)$ depending on whether zero, one, or two of these points satisfy the LTF. For any such LTF whose behavior we have specified above over the four points $(0, 0, 0), (0, 0, 1), (0, 1, 0), (0, 1, 1)$, there are five ways that the LTF can label the four points $(1, 0, 0), \dots, (1, 1, 1)$ depending on whether zero, \dots , or five of these points satisfy the LTF; and so on.

Next time we'll present a few more basic facts about LTFs; – this will conclude our preliminary discussion of these concept classes – and we'll discuss our first learning model (the Online Mistake Bound Model) and algorithms in that model.