Advanced Course in Machine Learning Spring 2010

Learning DNF Formulas

Handouts are jointly prepared by Shie Mannor and Shai Shalev-Shwartz

Much of computational learning theory deals with an algorithmic problem: Given random labeled examples from an unknown function $f : \{0,1\}^n \to \{0,1\}$, try to find a hypothesis $h : \{0,1\}^n \to \{0,1\}$ which is good at predicting the label of future examples.

Returning to the classical PAC model of Valiant [84], we identify a learning problem with a "concept class" C, which is a set of functions ("concepts") $f : \{0,1\}^n \to \{0,1\}$. Nature/adversary chooses one particular $f \in C$ and a probability distribution on inputs D. The learning algorithm now takes as inputs ϵ and δ, and also gets random examples $\langle x, f(x)\rangle$ with x drawn from D.

The goal: with probability $1-\delta$, output a hypothesis h which satisfies $\Pr_{x\sim D}[h(x) \neq f(x)] < \epsilon$.

The emphasis in computational learning theory is on efficiency: running time of algorithm, counting time 1 for each example, is hopefully $poly(n, 1/\epsilon, 1/\delta)$.

For example consider the problem of learning the concept class of conjunctions, i.e., C is the set of all AND functions. The following algorithm is due to Valiant [84].

- Start with the hypothesis $h = x_1 \wedge x_2 \wedge \ldots \wedge x_n$
- draw $O((n/\epsilon^2)\log(1/\delta))$ examples: whenever you see a positive example; e.g., $\langle 11010110, 1\rangle$, you know that the zero coordinates (in this case, x_3, x_5, x_8) cant be in the target AND; delete them from the hypothesis

It is not hard to show this algorithm actually works (exercise).

Probably the most important concept class we would like to learn is DNF formulas: e.g., the set of all functions like

$$f = (x_1 \wedge \overline{x_2} \wedge x_6) \vee (\overline{x_1} \wedge x_3) \vee (\overline{x_4} \wedge x_5 \wedge x_7 \wedge x_8).$$

(We actually mean poly-sized DNF: the number of terms should be $n^{O(1)}$, where n is the number of variables.)

Why so important?

- Natural form of knowledge representation for people
- Historical reasons: considered by Valiant, who called the problem "tantalizing" and "apparently [simple]" yet has proved a great challenge over the last 25 years
- Useful for machine learning - e.g., learning decision trees or decision lists.

Before we dive in, let me emphasize the problem with the PAC learning model: PAC-learning DNF formulas appears to be very hard. The fastest known algorithm, due to Klivans and Servedio, runs in time $\exp(n^{1/3}\log^2 n)$.

Technique: They show that for any DNF formula, there is a polynomial in $x_1, \ldots, x_n$ of degree at most $n^{1/3}\log n$ which is positive whenever the DNF is true and negative whenever the DNF is false. Linear programming can be used to find a hypothesis consistent with every example in time $\exp(n^{1/3}\log^2 n)$.

The more natural model, more difficult than PAC, in which the learner is forced to output a hypothesis which itself is a DNF. In this case, the problem is NP-hard.

0.1 Uniform Distribution

In 1990, Verbeugt [Ver90] observed that, under the uniform distribution, any term in the target DNF which is longer than $\log(n/\epsilon)$ is essentially always false, and thus irrelevant. This fairly easily leads to an algorithm for learning DNF under uniform in quasipolynomial time: roughly $n^{\log n}$.

In the next section we will see a powerful and sophisticated method of learning under the uniform distribution based on Fourier analysis. Fourier analysis proved very important for subsequent learning of DNF under the uniform distribution.

1 Fourier Analysis

We are interested in approximating and representing Boolean functions of the form $f : \{0,1\}^n \to \{-1,1\}$. Let $S \subseteq \{1,2,\ldots,n\}$ be a set of indices and let $\chi_S : \{0,1\}^n \to \{-1,1\}$ be a parity function (i.e., a XOR of all indices in S). Then:

$$f(x) = \sum_{S\subseteq\{1,2,\ldots,n\}} \hat{f}(S)\chi_S(x), \quad \forall x.$$

The coefficients $\hat{f}(S)$ are known as the "Fourier" coefficients. (Such coefficients exist because the parity functions form a basis.)

The inner product in this function space can now be defined as:

$$\langle f, g\rangle = \frac{1}{2^n}\sum_{x\in\{0,1\}^n} f(x)g(x),$$

with induced norm: $\|f\| = \sqrt{\langle f, f\rangle}$.

It is not hard to show that the $2^n - 1$ basis functions χ_S form an orthonormal basis. We therefore have that:

$$\hat{f}(S) = \langle f, \chi_S\rangle = \mathbb{E}_{x\sim\{0,1\}^n}[f(x)\chi_S(x)].$$

where the expectation is w.r.t. the uniform distribution over x.

We can easily write an expression for the covariance of two Boolean functions:

$$\begin{aligned}
\mathbb{E}_x[f(x)g(x)] &= \mathbb{E}_x\left[\left(\sum_{S_1}\hat{f}(S_1)\chi_{S_1}\right)\left(\sum_{S_2}\hat{f}(S_2)\chi_{S_2}\right)\right] \\
&= \mathbb{E}_x\left[\sum_{S_1,S_2}\hat{f}(S_1)\hat{f}(S_2)\chi_{S_1}\chi_{S_2}\right] \\
&= \sum_S \hat{f}(S_1)\hat{f}(S_2)\,\mathbb{E}_x[\chi_S^2] \\
&= \sum_S \hat{f}(S_1)\hat{f}(S_2),
\end{aligned}$$

where we used the orthonormality of the basis.

We obtain using the above result the well known Persival's Theorem:

$$\|f\|^2 = \mathbb{E}_x[f^2(x)] = \sum_S \hat{f}^2(s) = 1.$$

We now show by example one of the main features of Fourier analysis. Suppose we are looking for the Fourier representation of an AND function. So let us consider the AND of elements in T for some

$T \subset \{1, 2, \ldots, n\}$. Suppose we start from $T = \{1, 2, \ldots, 8\}$ for the sake of discussion.

$$\begin{aligned}\hat{f}(\{x_1, x_2, \ldots, x_9\}) &= \frac{1}{2^n} \sum_{x \in \{0,1\}^n} AND(x_1, \ldots, x_9)\chi_{\{x_1,\ldots,x_9\}} \\ &= \frac{1}{2^n} \sum_{x \in \{0,1\}^8 \times \{0,1\}^{n-9}} AND(x_1, \ldots, x_9)(\chi_{\{x_1,\ldots,0\}} + \chi_{\{x_1,\ldots,1\}}) = 0.\end{aligned}$$

A similar proof shows that if $T \subset S$ we have that $\hat{f}(S) = 0$. So we only need to consider $S \subset T$. Representing AND by:

$$AND(x_1, \ldots, x_8) = \prod_{i=1}^{8} \frac{1 - \chi_{\{x_i\}}}{2}$$

we obtain that $\hat{f}(s) = -1^{|S|}/2^8$ if $S \subset T$ and 0 if $T \subset S$.

1.1 Approximating a function by its spectrum

Suppose that a function f has a few large Fourier coefficients. In that case, getting a good approximation is possible as f can be approximated efficiently.

Suppose that f is a Boolean function and g is defined as:

$$g(x) = \sum_{t \subset T} \hat{f}(t)\chi_t(x)$$

Lemma 1 *Take $h = sign(g)$. Then the following inequality holds:*

$$\mathbb{E}_x[f(x) \neq h(x)] \leq \sum_{S \subset \{1,2,\ldots n\}} (\hat{f}(S) - \hat{g}(S))^2.$$

Proof Let I be the indicator function, i.e., $I(f(x) \neq h(x)) = 1$ and $I(f(x) = h(x)) = 0$. Then $I(f(x) \neq h(x)) \leq |f(x) - g(x)|$. So that

$$I(f(x) \neq h(x)) \leq |f(x) - g(x)|^2.$$

Taking the expectation:

$$\mathbb{E}\left[I(f(x) \neq h(x))\right] \leq \mathbb{E}[|f(x) - g(x)|^2].$$

Which means:

$$\begin{aligned}\Pr{}_x(f(x) \neq h(x)) &\leq \mathbb{E}[|f(x) - g(x)|^2] \\ &\leq \sum_{S \subset \{1,2,\ldots n\}} (\hat{f}(S) - \hat{g}(S))^2.\end{aligned}$$

■

In particular, if the set T contains most of the coefficients, we have a good quality approximation.

In light of the above, it would be simpler to say that a Boolean function $f(x)$ is ϵ approximated by another Boolean function $g(x)$ if the squared expected error $\mathbb{E}[(f(x) - g(x))^2] \leq \epsilon$.

1.2 Computing the Fourier coefficients from samples

A trivial algorithm is to sample $x_1, \ldots, x_m$ and output

$$a_s = \frac{1}{m}\sum_{i=1}^{m} f(x_i)\chi_S(x_i)$$

as the estimate of $\langle f, \chi_S\rangle$ or $\mathbb{E}_x[f\chi_S]$. (This is done independently for every S.)

An immediate application of the Chernoff bound yields the following corolary.

Corollary 1 *If $m > 2\ln(2/\delta)/\lambda^2$ then a_s is within λ of $\hat{f}(s)$ with probability $1-\delta$.*

We proceed to propose two approximations under additional assumptions on f.

1.2.1 Low degree f

We say that f has (α, d) degree if

$$\sum_{|S|\le d} \hat{f}^2(S) \ge 1-\alpha.$$

If a function has (α, d) degree there exists an $n^{O(d)}$ algorithm for learning f with accuracy of at least $1-\alpha$. Basically, we can enumerate over the subsamples and sample. (We omit some details here as this is not the focus of the lecture.)

1.2.2 Sprase f

We say that f is t-sparse if f has at most t non-zero Fourier coefficients. We will show below how to learn such functions using membership queries (but still assume we are under the uniform distribution).

2 Learning sparse functions

We first reduce the problem of finding a sparse function that closely approximates the target function f to finding all the large Fourier coefficients of f. We then describe an algorithm that finds these coefficients in polynomial time with probability $1-\delta$ of success. Then we show that if a function has a small L_1 norm, it can be approximated by sparse function and polynomial sized decision tree is such a function. Therefore, we can learn a function that approximates a decision tree in polynomial time with probability $1-\delta$.

Theorem 1 *If f is ϵ approximated by a t-sparse function g, then there exists a t-sparse function h such that: (1) $|\hat{h}(s)| \ge \epsilon/t$ for all the Fourier coefficients of h, and (2) f is $\epsilon + \epsilon^2/t$ approximated by h*

Proof Assume w.l.o.g. that $\hat{g}(s_i) = \hat{f}(s_i)$ for the t non-zero coefficient of $\hat{g}$. Let $A = \{s : \hat{g}(s) \neq 0\}$ and let $A' = A \cap \{s|\hat{f}(s) \ge \epsilon/t\}$. We let $h = \sum_{s\in A'} \hat{f}(s)\chi_s$. Since f is ϵ approximated by g we have that:

$$\sum_{s\notin A} \hat{f}(s)^2 = \epsilon.$$

Each element in $A \setminus A'$ has a magnitude less than ϵ/t and $|A \setminus A'| \le t$ since g is t sparse. So that $\sum_{s\in A\setminus A'} \hat{f}(s)^2 > t(\epsilon/t)^2$. By construction:

$$\mathbb{E}[|f-h|^2] = \sum_{s\notin A} \hat{f}(s)^2 + \sum_{s\in A\setminus A'} \hat{f}(s)^2 = \epsilon + t(\epsilon/t)^2 = \epsilon + \epsilon^2/t.$$

■

The above theorem reduced the problem of finding t-sparse function to ϵ-approximate f into the problem of finding all coefficients of f that are greater than a threshold such that the t-sparse function $O(\epsilon)$ approximates f. We denote the threshold by θ.

We now consider another useful property of Fourier spectrum. For a fixed string α of length k we define:

$$f_\alpha(x) = \sum_{z \in \{0,1\}^{n-k}} \hat{f}(\alpha z)\chi_z(x),$$

where αz is the concatenation of the strings α and z. The function f_α can be thought of as the spectrum when α is fixed.

The following result shows that it is possible to estimate f_α efficiently.

Theorem 2 *For any function f, $1 \leq k \leq n-1$, $\alpha \in \{0,1\}^k$, $x \in \{0,1\}^{n-k}$ and $y \in \{0,1\}^k$,*

$$f_\alpha(x) = \mathbb{E}_y[f(yx)\chi_\alpha(y)]$$

Proof The proof is left as an exercise. ■

Before providing the algorithm for finding sparse approximations we give two needed lemmas:

Lemma 2 *At most $1/\theta^2$ values of z satisfy $|\hat{f}(z)| \geq \theta$ for $0 < \theta < 1$.*

Proof From Parseval's Theorem $\sum_z \hat{f}(z)^2 = \mathbb{E}[f^2] = 1$. So that at most $1/\theta^2$ values of z can satisfy $|\hat{f}(z)| \geq \theta$. ■

Lemma 3 *At most $1/\theta^2$ functions f_α satisfy $\mathbb{E}[f_\alpha^2] \geq \theta^2$ for $\alpha \in \{0,1\}^k$ ($1 \leq k < n$).*

Proof From the definition of f_α we have:

$$\begin{aligned}\mathbb{E}[f_\alpha(x)^2] &= \mathbb{E}\Big[\sum_{z_1 \in \{0,1\}^{n-k}} \hat{f}(\alpha z_1)\chi_{z_1}(x) \sum_{z_2 \in \{0,1\}^{n-k}} \hat{f}(\alpha z_2)\chi_{z_2}(x)\Big] \\ &= \sum_{z_1 \in \{0,1\}^{n-k}} \sum_{z_2 \in \{0,1\}^{n-k}} \hat{f}(\alpha z_1)\hat{f}(\alpha z_2)\,\mathbb{E}[\chi_{z_1}(x)\chi_{z_2}(x)] \\ &= \sum_{z_1 \in \{0,1\}^{n-k}} \hat{f}(\alpha z_1)^2.\end{aligned}$$

Therefore, if $\hat{f}(\alpha z_1) > \theta$ for some $z_1 \in \{0,1\}^{n-k}$ it follows that $\mathbb{E}[f_\alpha^2] > \theta^2$ by Paseval's Theorem. The conclusion follows as before: there are at most $1/\theta^2$ value of α for which $\mathbb{E}[f_\alpha^2] \geq \theta^2$. ■

We can now introduce an algorithm due to Kushilevitz and Mansour to find coefficients $\hat{f}(z)$ such that $|\hat{f}(z)| \geq \theta$.

```
KM($\alpha$)
   if length ($\alpha$) $= n$ then
     output $\alpha$
  else if $\mathbb{E}[f_\alpha^2] \geq \theta^2$ then
     KM($\alpha 0$)
     KM($\alpha 1$)
  else output previous $\alpha$
```

The KM algorithm is first run with $KM(\emptyset)$ and then increases the length of α by growing its suffix as long as $|\hat{f}(s)| \geq \theta$. The argument in Lemma 3 shows that if $|\hat{f}(\alpha z)| \geq \theta$ for some $z \in \{0,1\}^{n-k}$, then $\mathbb{E}[f_\alpha^2] \geq \theta^2$. Therefore the condition $\mathbb{E}[f_\alpha^2] \geq \theta^2$ insures that the right coefficients will be output. Since at most $1/\theta^2$ of the f_α satisfy $\mathbb{E}[f_\alpha^2] \geq \theta^2$, the algorithm will make at most $2n/\theta$ recursive calls.

Approximating $\mathbb{E}[f_\alpha^2]$.
The KM algorithm above assumes that we can compute $\mathbb{E}[f_\alpha^2]$ in $O(1)$. In general we have to use sampling for that. We know we can sample $f_\alpha = \mathbb{E}_y[f(yx)\chi_\alpha(y)]$ and then sample again to estimate $\mathbb{E}[f_\alpha^2]$. A standard application of Chernoff bound leads to a number of samples that is polynomial in $1/\theta$, $\log n$ and $\log 1/\delta$. We cite the theorem from Kushilevitch and Mansour:

Theorem 3 *There exists a randomized algorithm that outputs a list of strings $\alpha_i \in \{0,1\}^n$ for any Boolean function $f : \{0,1\}^n \to \{-1,1\}$, any δ and θ, where $0 < \delta, \theta < 1$, such that*

1. *with probability $1-\delta$ the algorithm outputs $|\hat{f}(\alpha_i) \geq \theta|$ for every string α_i in the output list and there is no strings α for which $|f(\alpha)| \leq \theta/2$. The maximum length of the list is $4/\theta$; and*
2. *the algorithm runs in time polynomial in n, $1/\theta$, and $\log 1/\delta$.*

Applying Theorem 1, if f can be ϵ-approximated by a t-sparse function, then we need to find all the coefficients larger than $\theta = \epsilon/t$ to $O(\epsilon)$-approximate f. The running time of the algorithm will then a polynomial in n, t, $1/\theta$, and $\log 1/\delta$.

2.1 Function with small L_1 norm

Define the L_1 norm of a Boolean function f (also written as $L_1(f)$) to be $\sum_s |\hat{f}(s)|$. The following lemma shows that functions with small L_1 norm can be approximated by sparse functions.

Lemma 4 *For any Boolean function f, there exists a Boolean function h such that h is $L_1(f)^2/\epsilon$-sparse and $\mathbb{E}[(f-h)^2] \leq \epsilon$.*

Proof Consider the set $A = \{s : |\hat{f}(s)| \geq \epsilon/L_1(f)\}$. There are at most $L_1(f)/(\epsilon/L_1(f)) = L_1(f)^2/\epsilon$ elements in A. Let $h = \sum_{s\in A} \hat{f}(s)\chi_s(x)$, which is $L_1(f)^2/\epsilon$-sparse. Then,

$$\begin{aligned}\mathbb{E}[(f-h)^2] &= \sum_{s\notin A} \hat{f}(s)^2 \\ &\leq \max_{s\notin A} |\hat{f}(s)| \sum_s |\hat{f}(s)| \\ &\leq \frac{\epsilon}{L_1(f)} L_1(f) = \epsilon.\end{aligned}$$

■

.

From this lemma and Theorem 3 we can obtain the following conclusion:

Corollary 2 *If a Boolean function f has L_1 norm that is polynomial in n, then a function h that ϵ-approximates f is learnable in polynomial time (in n, $1/\epsilon$, and $\log 1/\delta$) with probability $1-\delta$.*

3 Approximating Decision Trees with Queries

If we can bound the L_1 norm of decision trees to some polynomial of n, then from Corollary 2, we can learn a function that approximates the decision tree in polynomial time $1-\delta$ of the time.

Consider decision tree with m leafs, where m is polynomial in n. It follows that the size of the tree is also polynomial in n. Define the function $AND_{b_1,\dots,b_d}$ as the and of $b_1, b_2, \dots, b_d$. We are going to find $L_1(AND_{b_1,\dots,b_d})$. We start with the following observations for bounding the L_1 norm of functions:

Lemma 5 $L_1(f+g) \leq L_1(f) + L_2(g)$.

Lemma 6 $L_1(f \cdot g) \leq L_1(f) \cdot L_2(g)$.

Writing

$$AND_{b_1,\dots,b_d}(x_1, x_2, \dots, x_d) = \prod_{i=1}^{d} \frac{1 + (-1)^{b_i}\chi_i(x)}{2},$$

where $\chi_i(x) = 1$ if $x_i = 0$ and -1 if $x_i = 1$. From Lemma 6 we have that:

$$L_1(AND_{b_1,\dots,b_d}) \leq \prod_{i=1}^{d} L_1\left(\frac{1 + (-1)^{b_i}\chi_i(x)}{2}\right) \leq 1.$$

But on the other hand since $AND_{b_1,\dots,b_d}(b_1, \dots, b_d) = 1$ (we have a value of 1 for the assignment of ones) we must have $L_1(AND_{b_1,\dots,b_d}) \geq 1$ so that $L_1(AND_{b_1,\dots,b_d}) = 1$.

We now consider a decision tree of depth d. A path of length $d-1$ from the root node to a leaf can be expressed as $b = (b_1, \dots, b_d)$, where $b_1, \dots, b_d$ are the assignment of the variables along the path. Therefore, if $AND_{b_1,\dots,b_d}(x_1, \dots, x_d) = 1$, the decision tree will output the value of the leaf. A decision tree $DT : \{0,1\}^d \to \{0,1\}$ can therefore be expressed as the sum of all the paths that returns 1 at the leaf. Let this set of paths be P. Since there are m leafs, $|P| \leq m$. By Lemma 5 and since $L_1(AND) = 1$, we have

$$L1(DT) = L_1(\sum_{b \in P} AND_b) \leq m.$$

We conclude that the L_1 norm of a decision tree with m leaves is at most m, which is polynomial in n. By Corollary 2, polynomial sized decision tree can be approximated in polynomial time with queries with probability $1 - \delta$.

4 Conclusion

A still open problem in DNF learning is whether DNF can be learned in polynomial time under the uniform distribution. One might ask, What is the current stumbling block? Why can't we seem to do better than $n^{\log n}$ time? We are stuck even on what seems like a much simpler problem: the junta-learning problem.

A k-junta is a function on n bits which happens to depend on only k bits. (All other $n - k$ coordinates are irrelevant.)

Since every boolean function on k bits has a DNF of size 2^k, it follows that the set of all $\log n$-juntas is a subset of the set of all polynomial-size DNF formulas. Thus to learn DNF under uniform in polynomial time, we must be able to learn $\log(n)$-juntas under uniform in polynomial time.

There is an extremely naive algorithm running in time n^k essentially, test all possible sets of k variables to see if they are the junta. Much work has to be spent to give an algorithm for learning k-juntas under the uniform distribution which runs in time $n^{.704k}$. The technique involves trading off different polynomial representations of boolean functions (Mossel et al, 03). This is not much of an improvement for the important case of $k = \log n$. However at least it we know that n^k can be improved.

The open problem (cash prize!) called the junta-learning problem is described below and seems like a major step on the way of solving the DNF learning problem.

An unknown and arbitrary function $f : \{0,1\}^n \to \{0,1\}$ is selected, which depends only on some k of the bits. The algorithm gets access to uniformly randomly chosen examples, $\langle x, f(x) \rangle$. With probability $1 - \delta$ the algorithm should output at least one bit (equivalently, all k bits) upon which f depends. The algorithms running time is considered to be of the form $n^{\alpha} poly(n, 2k, 1/\delta)$, and α is the important measure of complexity. Our goal is to get $\alpha << k$.