Machine Learning: Foundations **Fall Semester, 2010/2011**

Lecture 14: Fourier Transform

Lecturer: Yishay Mansour *Scribes: Avi Goren & Oron Navon* [1]

14.1 Introduction

In this lecture, we examine the Fourier Transform representation of functions and show how it can be used to learn such functions by approximating their Fourier coefficients. We will demonstrate the use of this approach to learn functions on boolean variables, such as decision trees, but note that some of its concepts can be extended to nonboolean functions as well.

14.2 Model and Definitions

The learning model has a class of functions $\mathcal{F}$ which we wish to learn. Out of this class there is a specific function $f \in \mathcal{F}$ which is chosen as a target function. A learning algorithm has access to examples. An example is a pair $< x, f(x) >$, where x is an input and $f(x)$ is the value of the target function on the input x. After requesting a finite number of examples, the learning algorithm outputs a hypothesis h. The error of a hypothesis h, with respect to the function f, is defined to be $error(f,h) \stackrel{\triangle}{=} \Pr[f(x) \neq h(x)]$, where x is distributed uniformly over $\{0,1\}^n$.

We discuss two models for accessing the examples. In the *uniform distribution model* the algorithm has access to a random source of examples. Each time the algorithm requests an example, a random input $x \in \{0,1\}^n$ is chosen uniformly, and the example $< x, f(x) >$ is returned to the algorithm. In the *membership queries model*, the algorithm can query the unknown function f on any input $x \in \{0,1\}^n$ and receive the example $< x, f(x) >$.

A randomized algorithm A *learns* a class of functions $\mathcal{F}$ if for every $f \in \mathcal{F}$ and $\varepsilon, \delta > 0$ the algorithm outputs an hypothesis h such that with probability at least $1 - \delta$,

$$error(f,h) \leq \varepsilon$$

The algorithm A learns in *polynomial time* if its running time is polynomial in n, $1/\varepsilon$, and $\log 1/\delta$.

[1]based on the following publication: Yishay Mansour, Learning Boolean Functions via the Fourier Transform, In *Theoretical Advances in Neural Computation and Learning*, (V.P. Roychodhury and K-Y. Siu and A. Orlitsky, ed.), 391–424 (1994).

The functions we are interested in have boolean inputs and are of the form,

$$f : \{0,1\}^n \to \mathbb{R}.$$

We are mainly interested in boolean functions of the form,

$$f : \{0,1\}^n \to \{-1,+1\}.$$

We are interested in creating a basis for those functions. Recall that a basis, in this case, is a set of basis functions such that any function of the form $f : \{0,1\}^n \to \mathbb{R}$ can be represented as a linear combination of the basis functions. One basis is the functions $term_\alpha(x)$, for $\alpha \in \{0,1\}^n$, where $term_\alpha(\alpha) = 1$ and $term_\alpha(\beta) = 0$ for $\beta \neq \alpha$. Any function f can be written as $\sum_\alpha a_\alpha term_\alpha(x)$ where the constants are $a_\alpha = f(\alpha)$. In the following we describe a different basis which is called the Fourier basis.

The Fourier basis has 2^n functions; for each $\alpha \in \{0,1\}^n$ there is a function $\chi_\alpha : \{0,1\}^n \to \{+1,-1\}$. The value of a basis function χ_α is,

$$\chi_\alpha(x) = (-1)^{\sum_{i=1}^n x_i \alpha_i}.$$

An alternative way of defining the same functions, which we also use throughout the text, is to denote the basis functions using a subset $S \subseteq \{1,\ldots,n\}$. The set S defines the set of inputs on which the function χ_S is defined. The value of χ_S depends on the parity of the inputs in S. Formally,

$$\chi_S(x) = \prod_{i \in S} (-1)^{x_i} = \begin{cases} +1 & \text{if } \sum_{i\in S} x_i \bmod 2 = 0 \\ -1 & \text{if } \sum_{i\in S} x_i \bmod 2 = 1 \end{cases}$$

Note that,

$$\chi_S(x) \equiv \chi_\alpha(x)$$

where $S \subseteq \{1,\ldots,n\}$, and $i \in S \iff \alpha_i = 1$.

The *inner product* of two functions f and g is,

$$< f, g > = \frac{1}{2^n} \sum_{x \in \{0,1\}^n} f(x) g(x) \; = \; E[f \cdot g],$$

where E is the expected value of $f \cdot g$ using the uniform distribution on $\{0,1\}^n$. The *norm* of a function is $\|f\| = \sqrt{< f, f >}$.

14.2.1 Basis Properties

We mention a few properties of the Fourier basis defined above.

- The basis is *normal*, i.e. $\|\chi_S\| \equiv 1$, since $\forall x : \chi_S^2(x) = 1$.
- $\chi_\alpha \cdot \chi_\beta = \chi_{\alpha \oplus \beta}$, where the addition is bit-wise exclusive or.
- If $\alpha \neq 0$ then $E[\chi_\alpha] = 0$.
- *Orthogonality.* $< \chi_\alpha, \chi_\beta >= 0 \Leftrightarrow \alpha \neq \beta$, due to the fact that

$$< \chi_\alpha, \chi_\beta >= E[\chi_\alpha \cdot \chi_\beta] = E[\chi_{\alpha\oplus\beta}] = \begin{cases} 1 & \text{if } \alpha = \beta \\ 0 & \text{if } \alpha \neq \beta \end{cases}.$$

- *Dimensionality.* The orthogonality implies that the dimension of the basis is 2^n.

From the dimensionality of the basis we can deduce that every function $f : \{0,1\}^n \to \mathbb{R}$ can be represented as a linear combination of basis functions.

Claim 14.1 *For any $f : \{0,1\}^n \to \mathbb{R}$ then,*

$$f(x) = \sum_{\alpha \in \{0,1\}^n} a_\alpha \chi_\alpha(x),$$

where $a_\alpha = \hat{f}(\alpha) =< f, \chi_\alpha >$.

Parseval's identity relates the values of the coefficients to the values of the function.

Theorem 14.2 (Parseval's Identity) *For any $f : \{0,1\}^n \to \mathbb{R}$,*

$$\sum_{\alpha \in \{0,1\}^n} \hat{f}^2(\alpha) = E[f^2]$$

Proof: Consider the following simple algebraic manipulations.

$$\begin{aligned} E_x[f^2(x)] &= E_x\left[\left(\sum_\alpha \hat{f}(\alpha)\chi_\alpha(x)\right)\left(\sum_\beta \hat{f}(\beta)\chi_\beta(x)\right)\right] \\ &= \sum_\alpha \sum_\beta \hat{f}(\alpha)\hat{f}(\beta) E_x\left[\chi_{\alpha\oplus\beta}(x)\right] \end{aligned}$$

If $\alpha \neq \beta$, then $E_x[\chi_{\alpha\oplus\beta}(x)]$ is zero,, therefore, the expression reduces to,

$$E[f^2] = \sum_\alpha \hat{f}^2(\alpha),$$

which completes the proof. ■

Parseval's identity for *boolean* functions, i.e. $f : \{0,1\}^n \to \{-1,+1\}$, states that $\sum_\alpha \hat{f}^2(\alpha) \equiv 1$

14.2.2 Example: Fourier transform of AND

We compute the Fourier transform of the following function:

$$AND\,(x_1,\ldots,x_n)=\prod_{i=1}^{n}x_i,$$

where $x_i\in\{0,1\}$. We can write

$$x_i=\frac{1-\chi_{\{i\}}(\vec{x})}{2}$$

And over all the x_i's, we get

$$AND(x_1,\ldots,x_n)=\prod_{i=1}^{n}\frac{1-\chi_{\{i\}}(\vec{x})}{2}=2^{-n}\cdot\sum_{S\subseteq[1\ldots n]}1^{n-|S|}\prod_{i\in S}(-\chi_{\{i\}}(\vec{x}))=2^{-n}\cdot\sum_{S}(-1)^{|S|}\chi_S(\vec{x}).$$

Therefore, the Fourier coefficient of set S is,

$$AN\hat{D}(S)=2^{-n}\cdot(-1)^{|S|}.$$

14.3 Learning and Fourier Transform

We start by considering an example. Let f be a boolean function, such that for some (known) β it holds that $\hat{f}(\beta)=0.9$, and no other information about f is known. A natural hypothesis would be, $h(x)\equiv 0.9\chi_\beta(x)$, and we would like to estimate the error squared for h. (Intuitively it is clear that if the expected error squared is small then we have a good estimation in some sense. Later we will show how this parameter relates to boolean prediction.)

Let the error function be $error_h(x)=|f(x)-h(x)|$. The expected error square is,

$$E[(f-h)^2]=\left(\sum_{\alpha\neq\beta}\hat{f}^2(\alpha)\right)+\left(\hat{f}^2(\beta)-\hat{f}^2(\beta)\right)=1-\hat{f}^2(\beta)=1-0.81=0.19.$$

Introducing another piece of information, e.g. $\hat{f}(\gamma)=0.3$, would reduce the error. Our new hypothesis would be $h(x)\equiv 0.9\chi_\beta(x)+0.3\chi_\gamma(x)$, and the expected error square is,

$$E[(f-h)^2]=1-\hat{f}^2(\beta)-\hat{f}^2(\gamma)=1-0.81-0.09=0.1.$$

Boolean Prediction

In the example above, our hypothesis $h(x)$ was not a boolean function. In order to get a boolean prediction we can output $+1$ if $h(x)\geq 0$ and -1 if $h(x)<0$. More formally,

Definition The *Sign* function takes a real parameter and return its sign,

$$Sign(z)\stackrel{\text{def}}{=}\begin{cases}+1 & \text{if } z\geq 0\\ -1 & \text{if } z<0\end{cases}.$$

The following claim shows that the expected error squared bound the probability of an error in predicting according to the sign of h.

Claim 14.3 *If f is a Boolean function then,*

$$\Pr[f(x) \neq Sign(h(x))] \leq E[(f-h)^2].$$

Proof: Let I be the indicator function, i.e.,

$$I\left[f(x) \neq Sign(h(x))\right] \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } f(x) \neq Sign(h(x)) \\ 0 & \text{if } f(x) = Sign(h(x)). \end{cases}$$

The probability of error is,

$$\Pr[f(x) \neq Sign(h(x))] = \frac{1}{2^n} \sum_x I\left[f(x) \neq Sign(h(x))\right].$$

We show that for every $x \in \{0,1\}^n$, $I[f(x) \neq Sign(h(x))] \leq (f(x)-h(x))^2$, which implies the claim. We consider the following two cases.

- If $f(x) = Sign(h(x))$ then $I\left[f(x) \neq Sign(h(x))\right] = 0$, so clearly,

$$I\left[f(x) \neq Sign(h(x))\right] = 0 \leq (f(x)-h(x))^2.$$

- If $f(x) \neq Sign(h(x))$ then we have $|f(x)-h(x)| \geq 1$. Therefore,

$$I\left[f(x) \neq Sign(h(x))\right] = 1 \leq (f(x)-h(x))^2.$$

■

As a result from the claim we can use $E[(f-h)^2]$ as an upper bound for $\Pr[f(x) \neq Sign(h(x))]$. Notice that the above proof holds for any distribution although we apply it here only to the uniform distribution.

The following definition would be useful.

Definition 14.4 *A (real) function g ε-*approximates *f if $E[(f(x)-g(x))^2] \leq \varepsilon$.*

14.3.1 Approximating a single coefficient

Recall the example in which we "know" that the coefficient at β is "large". There we assumed that we are given the value of the coefficient of β (i.e. $\hat{f}(\beta)$ is given). In this section we show how to approximate it from random examples.

We are interested in approximating a coefficient $\hat{f}(\beta)$ for a given β. Recall that,

$$\hat{f}(\beta) =< f, \chi_\beta >= E[f \cdot \chi_\beta]$$

Since we are interested only in an estimate, we can sample randomly x_i's and take the average value. The sampling is done by choosing the x_is from the uniform distribution, and the estimate is,

$$a_\beta = \frac{1}{m} \sum_{i=1}^{m} f(x_i)\chi_\beta(x_i).$$

Using the Chernoff bounds, for $m \geq \frac{2}{\lambda^2} \ln\left(\frac{2}{\delta}\right)$, the probability that the error in the estimate is more than λ is,

$$\Pr[|\hat{f}(\beta) - a_\beta| \geq \lambda] \;\; \leq \;\; 2e^{-\lambda^2 m/2} \leq \delta.$$

Given that $|\hat{f}(\beta) - a_\beta| \leq \lambda$ then $(\hat{f}(\beta) - a_\beta)^2 \leq \lambda^2$, and

$$E[(f - a_\beta\chi_\beta)^2] = \sum_{\alpha \neq \beta} \hat{f}^2(\alpha) + (\hat{f}(\beta) - a_\beta)^2 \leq 1 - \hat{f}^2(\beta) + \lambda^2.$$

Recall that the original error was $1 - \hat{f}^2(\beta)$, given that we knew exactly the value of $\hat{f}(\beta)$. Hence, the "penalty" for estimating $\hat{f}(\beta)$ there is an additional error term of λ^2.

14.3.2 Low Degree Algorithm

In the previous section we showed how to approximate a single coefficient. For many classes of functions, each function can be approximated by considering only a small number of coefficients. Furthermore, those are the coefficients that correspond to small sets[2].

Assume f is defined *"mainly"* on the *"low"* coefficients. Formally, a function has an (α, d)-degree if $\sum_{S:|S|>d} \hat{f}^2(S) \leq \alpha$. The algorithm that approximates an (α, d)-degree function is the following.

- Sample m examples, $< x_i, f(x_i) >$. For each S, with $|S| \leq d$, compute $a_S = \frac{1}{m}\sum_{i=1}^{m} f(x_i)\chi_S(x_i)$, where $m \geq \frac{1}{2}\sqrt{\frac{n^d}{\varepsilon}} \cdot \ln\left(\frac{2n^d}{\delta}\right)$.

[2]We call the coefficients of small sets the "low" coefficients, and the coefficients of large sets the "high" coefficients.

- Output the function $h(x)$,
$$h(x) \stackrel{\text{def}}{=} \sum_{|S|\le d} a_S \chi_S(x).$$

Theorem 14.5 *Let f be an (α, d)-degree function. Then with probability $1-\delta$ the Low Degree Algorithm outputs a hypothesis h such that $E[(f-h)^2] \le \alpha + \epsilon$.*

Proof: First we claim that the algorithm approximates each coefficient within λ. More precisely,
$$\Pr[|a_S - \hat{f}(S)| \ge \lambda] \le 2e^{-\lambda^2 m/2}$$
The error of $h(x)$ is bounded by,
$$E[(f-h)^2] = \alpha + \sum_{|S|\le d} \left(\hat{f}(S) - a_S\right)^2 \le \alpha + \sum_{|S|\le d} \lambda^2 \le \alpha + n^d \cdot \lambda^2.$$
We want to bound the error by $\alpha + \varepsilon$. Therefore,
$$n^d \cdot \lambda^2 \le \varepsilon \Longrightarrow \lambda \le \sqrt{\frac{\varepsilon}{n^d}}.$$
This needs to hold with probability $1-\delta$, therefore,
$$2e^{-\lambda^2 m/2} \cdot n^d \le \delta,$$
which holds for
$$m \ge \frac{2n^d}{\varepsilon} \ln\left(\frac{2n^d}{\delta}\right).$$
■

Note that we did not "really" use the low degree in any other way than to bound the size of the set of the "interesting" coefficients. We can use the same algorithm to learn any function that can be approximated by using a small set of coefficients which is *known in advance* to the algorithm. (In a similar way we would approximate each coefficient in the set separately, and the time complexity would be proportional to the number of coefficients.)

14.3.3 Learning Sparse Functions

In the previous section we showed how to learn a function by approximating its Fourier coefficients on sets that are smaller than a certain parameter d, with a running time of $O(n^d)$. However, in many cases most of the coefficients of sets that are smaller than d may still be negligible. Furthermore, it might be the case that a function can be approximated by a small number of coefficients, but the coefficients do not correspond to small sets.

In this section we describe a learning algorithm that learns the target function by finding a sparse function, i.e. a function with a small number of non-zero coefficients. The main advantage of the algorithm is that its running time is polynomial in the number of non-zero coefficients. However, the disadvantage is that the algorithm uses the query model. We start by defining a sparse function.

Definition 14.6 *A function f is* t-sparse *if it has at most t non zero Fourier coefficients.*

The main result in this section is that if f can be ε-approximated by some polynomially-sparse function g then there is a randomized polynomial time algorithm that finds a polynomially sparse function h that $O(\varepsilon)$-approximates f. This algorithm works in the query model and the approximation is with respect to the uniform distribution.

The first step is to show that if f can be approximated by a polynomially sparse function g, it can be approximated by a polynomially sparse function that has only "significant" coefficients. We remark that we do not make a "direct" use of g (e.g., by approximating g instead of approximating f) but only use its existence in the analysis.

Lemma 14.7 *If f can be ε-approximated by a t-sparse function g then there exists a t-sparse function h such that $E[(f-h)^2] \leq \varepsilon + \varepsilon^2/t$ and all the non-zero coefficients of h are at least ε/t.*

Proof: Let $\Lambda = \{S | \hat{g}(S) \neq 0\}$ and $\Lambda' = \Lambda \cap \{S| \ \ |\hat{f}(S)| \geq \frac{\varepsilon}{t}\}$. Since g is t-sparse, then $|\Lambda'| \leq |\Lambda| \leq t$. Let h be the function obtained from Λ' by taking the respective coefficients of f. Namely,

$$h(x) = \sum_{S \in \Lambda'} \hat{f}(S)\chi_S(x).$$

Clearly h is t-sparse. We now bound the value of $E[(f-h)^2]$ as follows,

$$E_x[(f(x)-h(x))^2] = \sum_{S \notin \Lambda} \hat{f}^2(S) + \sum_{S \in \Lambda - \Lambda'} \hat{f}^2(S).$$

The first term bounds $E[(f-g)^2]$ from below, since it includes all the coefficients that are not in the support of g. We bound the second term using the fact that $|\Lambda - \Lambda'| \leq t$, and for each $S \in \Lambda - \Lambda'$, it holds that $|\hat{f}(S)| \leq \frac{\varepsilon}{t}$. Hence

$$E_x[(f(x)-h(x))^2] \leq \varepsilon + (\frac{\varepsilon}{t})^2 t = \varepsilon + \varepsilon^2/t,$$

which completes the proof. ■

The above lemma has reduced the problem of approximating f by a t-sparse function to the problem of finding all the coefficients of f that are greater than a threshold of ε/t. Note that the function h defined above does not necessarily contain all the coefficients of f

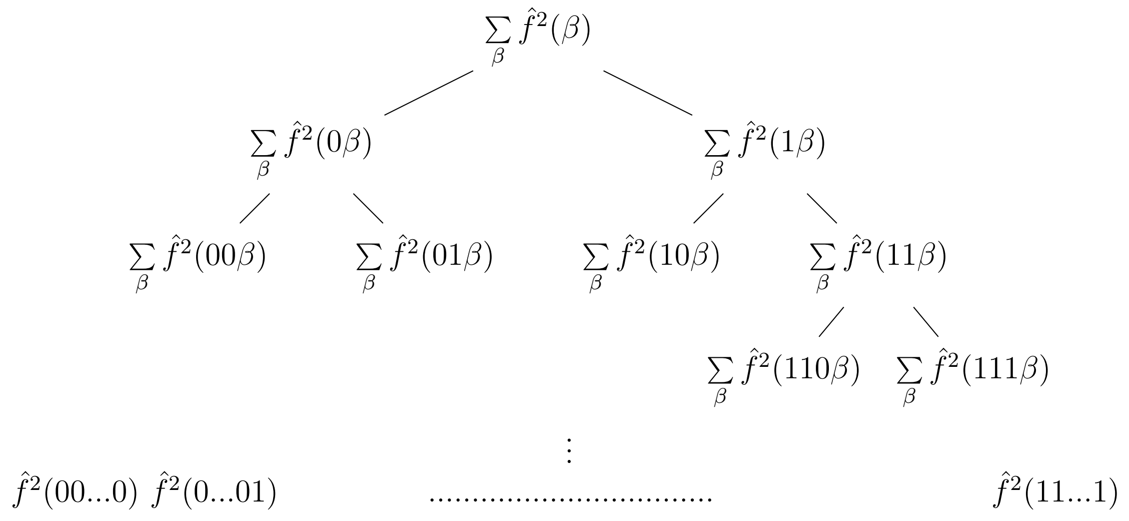
$\sum_\beta \hat{f}^2(\beta)$
$\sum_\beta \hat{f}^2(0\beta)$
$\sum_\beta \hat{f}^2(1\beta)$
$\sum_\beta \hat{f}^2(00\beta)$
$\sum_\beta \hat{f}^2(01\beta)$
$\sum_\beta \hat{f}^2(10\beta)$
$\sum_\beta \hat{f}^2(11\beta)$
$\sum_\beta \hat{f}^2(110\beta)$
$\sum_\beta \hat{f}^2(111\beta)$
$\hat{f}^2(00...0)$ $\hat{f}^2(0...01)$ .... $\hat{f}^2(11...1)$

Figure 14.1: Large Coefficients Search Tree

which are greater than ε/t, but only those which appear also in g. However, adding these coefficients to h can only make h a better approximation for f. In any case, the number of these coefficients, as follows from Lemma 14.10 below, cannot be too high.

Our aim is to show a randomized polynomial time procedure that given a function f and a threshold θ outputs (with probability $1-\delta$) all the coefficients for which $|\hat{f}(z)| \geq \theta$. The procedure runs in polynomial time in n, $1/\theta$ and $\log 1/\delta$. (Having f means that the algorithm can use *queries*.)

Let us consider the tree search structure shown in Figure 14.1. Our goal is to find the largest coefficients located in the leaves. This motivates us to explore relevant sub trees and eliminate sub trees with no coefficients (leaves) for which $|\hat{f}^2(z)| \geq \theta^2$. Hence, the algorithm partitions the coefficients according to their prefix. Let

$$f(x) = \sum_{z\in\{0,1\}^n} \hat{f}(z)\chi_z(x).$$

For every $\alpha \in \{0,1\}^k$, we define the function $f_\alpha : \{0,1\}^{n-k} \to \mathbb{R}$ as follows:

$$f_\alpha(x) \triangleq \sum_{\beta\in\{0,1\}^{n-k}} \hat{f}(\alpha\beta)\chi_\beta(x)$$

In other words, the function $f_\alpha(x)$ includes all the coefficients $\hat{f}(z)$ of f such that z starts with α and no other coefficient (i.e. all other coefficients are 0). This immediately gives the key idea for how to find the significant coefficients of f: find (recursively) the significant coefficients of f_0 and f_1. During the learning process we can only query for the value of the target function f in certain points. Therefore, we first have to show that $f_\alpha(x)$ can be efficiently computed using such queries to f. Actually, we do not need to compute the exact value of $f_\alpha(x)$ but rather it is sufficient to approximate it. The following lemma gives an equivalent formulation of f_α, which is computationally much more appealing:

Lemma 14.8 *Let f be a function, $1 \le k < n$, $\alpha \in \{0,1\}^k$ and $x \in \{0,1\}^{n-k}$, then*

$$f_\alpha(x) = E_{y\in\{0,1\}^k}[f(yx)\chi_\alpha(y)].$$

The above formulation implies that even though we cannot compute the value of $f_\alpha(x)$ exactly we can approximate it by approximating the above expectation.

Proof: Let $f(yx) = \sum_z \hat{f}(z)\chi_z(yx)$. Note that if $z = z_1z_2$, where $z_1 \in \{0,1\}^k$, then $\chi_z(yx) = \chi_{z_1}(y)\chi_{z_2}(x)$. Therefore,

$$\begin{aligned} E_y[f(yx)\chi_\alpha(y)] &= E_y\left[\left(\sum_{z_1}\sum_{z_2}\hat{f}(z_1z_2)\chi_{z_1}(y)\chi_{z_2}(x)\right)\chi_\alpha(y)\right] \\ &= \sum_{z_1}\sum_{z_2}\hat{f}(z_1z_2)\chi_{z_2}(x)E_y[\chi_{z_1}(y)\chi_\alpha(y)] \end{aligned}$$

where y and z_1 are strings in $\{0,1\}^k$ and z_2 is in $\{0,1\}^{n-k}$. By the orthonormality of the basis, it follows that $E_y[\chi_{z_1}(y)\chi_\alpha(y)]$ equals 0 if $z_1 \ne \alpha$, and equals 1 if $z_1 = \alpha$. Therefore, only the terms with $z_1 = \alpha$ contribute in the sum. Thus, it equals,

$$E_y[f(yx)\chi_\alpha(y)] = \sum_{z_2\in\{0,1\}^{n-k}} \hat{f}(\alpha z_2)\chi_{z_2}(x) = f_\alpha(x),$$

which completes the proof of the lemma. ■

Since both $|f(x)| = 1$ and $|\chi_\alpha(y)| = 1$ we derive the following corollary on the value of $f_\alpha(x)$.

Corollary 14.9 *Let f be a boolean function, $1 \le k < n$, $\alpha \in \{0,1\}^k$ and $x \in \{0,1\}^{n-k}$, then*

$$|f_\alpha(x)| \le 1.$$

We showed how to decompose a function f into functions f_α, $\alpha \in \{0,1\}^k$, such that each coefficient of f appears in a unique f_α. Recall that our aim is to find the coefficients $\hat{f}(z)$ such that $|\hat{f}(z)| \ge \theta$. The next lemma claims that this cannot hold for "too many" values of z, and that the property $E[f_\alpha^2] \ge \theta^2$ cannot hold for "many" α (of length k) simultaneously.

Lemma 14.10 *Let f be a* boolean *function, and $\theta > 0$. Then,*

1. *At most $1/\theta^2$ values of z satisfy $|\hat{f}(z)| \ge \theta$.*

```
SUBROUTINE SA($\alpha$)
        IF $E[f_\alpha^2] \geq \theta^2$ THEN
                IF $|\alpha| = n$ THEN OUTPUT $\alpha$
                        ELSE SA($\alpha 0$); SA($\alpha 1$);
```

Figure 14.2: Subroutine SA

2. *For any $1 \leq k < n$, at most $1/\theta^2$ functions f_α with $\alpha \in \{0,1\}^k$ satisfy $E[f_\alpha^2] \geq \theta^2$.*

Proof: By the assumption that f is a boolean function combined with Parseval's equality, we get

$$\sum_{z \in \{0,1\}^n} \hat{f}^2(z) = E[f^2] = 1.$$

Therefore, (1) immediately follows. Similarly, using the definition of f_α,

$$E[f_\alpha^2] = \sum_{\beta \in \{0,1\}^{n-k}} \hat{f}^2(\alpha\beta).$$

Thus, if $|\hat{f}(\alpha\beta)| \geq \theta$, for some $\beta \in \{0,1\}^{n-k}$, then $E[f_\alpha^2] \geq \theta^2$. By the above two equalities the following holds,

$$\sum_{\alpha \in \{0,1\}^k} E[f_\alpha^2] = E[f^2] = 1.$$

Therefore, at most $1/\theta^2$ functions f_α have $E[f_\alpha^2] \geq \theta^2$, which completes the proof of (2). ∎

The Sparse Algorithm

By now the algorithm for finding the significant coefficients of a function f should be rather obvious. It is described by the recursive subroutine SA, appearing in Figure 14.2. We start the algorithm by calling SA(λ), where λ is the empty string.

We know that each coefficient of f_α appears in exactly one of $f_{\alpha 0}$ and $f_{\alpha 1}$. Also, if $|\hat{f}(\alpha\beta)| \geq \theta$, for some $\beta \in \{0,1\}^{n-k}$, then $E[f_\alpha^2] \geq \theta^2$ (note that when $|\alpha| = n$, then $E[f_\alpha^2] = \hat{f}^2(\alpha)$). Therefore, the algorithm outputs all the coefficients that are larger than θ.

By Lemma 14.10, we also know that the number of α's for which $E[f_\alpha^2] \geq \theta^2$ is bounded by $1/\theta^2$, for each length of α. Thus, the total number of recursive calls is bounded by $O(n/\theta^2)$.

This is a sketch of the algorithm. Formally, we are still not done, since this algorithm assumes that we can compute $E[f_\alpha^2]$ exactly, something that is not achievable in polynomial time. On the other hand, we can approximate $E[f_\alpha^2] = E_{x \in \{0,1\}^{n-k}}[E^2_{y \in \{0,1\}^k}[f(yx)\chi_\alpha(y)]]$.

14.3.4 Decision Trees

A *Decision Tree* is a binary tree where each internal node is labeled with a variable, and each leaf is labeled with either $+1$ or -1. Each decision tree defines a boolean function as follows. An assignment to the variables determines a unique path from the root to a leaf: at each internal node the left (respectively right) edge to a child is taken if the variable named at that internal node is 0 (respectively 1) in the assignment. The value of the function at the assignment is the value at the leaf reached. The *depth* of a decision tree is the length of the longest path from the root to a leaf and denoted by $\texttt{DT-depth}(T)$. For a function f we denote by $\texttt{DT-depth}(f)$ the minimum depth of a decision tree that computes f. Note that every boolean function on n variables, can be represented by a decision tree with at most 2^n nodes and depth at most n.

We show that decision trees can be approximated by considering only the "low" coefficients. In this case the low coefficients are coefficients of set of at most logarithmic size in the number of nodes in the tree.

Claim 14.11 *Let T be a decision tree with m nodes. There exists a function h such that all the Fourier coefficients of h on sets larger than t are 0, where $t = \log m/\epsilon$, and $\Pr[T \neq h] \leq \epsilon$.*

Proof: Let h be the decision tree that results from truncating T at depth t. At the truncated places we add a leaf with value $+1$. The probability of reaching a specific replaced node is no more than ϵ/m. the number of replaced nodes is bounded by the number of leaves m so we get that reaching any replaced node is done with a probability no more than ϵ, therefore $\Pr[T \neq h] \leq \epsilon$.

We need to show that $\hat{h}(S) = 0$, for any S such that $|S| > t$. Since h is a decision tree, we can write a term for each leaf[3], and we are guaranteed that exactly one term is `true` for each input. Let $term_i$ be the terms, then we can write $h(x) = \sum_{i=1}^{m} \sigma_i term_i(x)$. Each term has at most t variables, hence, its largest non-zero coefficient is of a set of size at most t. In other words, for each $term_i(x)$, all the coefficients of sets larger than t are zero. Since h is a linear combination of such terms all its coefficients of sets larger than t are zero. ■

From the above claim we see that it is sufficient to approximate the coefficients of size at most t to approximate the decision tree. This can be done by running the Low Degree algorithm, which results in a running time of $O(n^{\log m/\varepsilon})$.

In what follows we show that in fact a decision tree can be approximated by a sparse function, where the number of coefficients is polynomial in the size of the decision tree.

First, we show that function with small L_1 (to be define latter) can be approximated by sparse functions. Later we show that decision trees are included in this class.

[3]The term will be the conjunction of the variables on the path from the root to the leaf (with the appropriate variables negated), such that the term is true iff the input reaches this leaf.

Learning Functions with the norm L_1

We start by defining the L_1 norm of a function.

Definition 14.12 *Let $L_1(f)$ be,*

$$L_1(f) = \sum_{S \subseteq \{1,\ldots,n\}} |\hat{f}(S)|.$$

The next Lemma shows that if $L_1(f)$ is "small" then f can be approximated by a "few" coefficients.

Theorem 14.13 *For every boolean function f there exists a function h, such that h is $\frac{(L_1(f))^2}{\epsilon}$-sparse and,*

$$E[(f-h)^2] \le \epsilon.$$

Proof: Let Λ be,

$$\Lambda = \{S \mid |\hat{f}(S)| \ge \frac{\epsilon}{L_1(f)}\}$$

Intuitively, Λ includes all the "significant" coefficients. From the bound of the sum of the coeffiecnts in absolute value (i.e. $L_1(f)$) it follows Λ size is bounded as follows,

$$|\Lambda| \le \frac{L_1(f)}{\epsilon / L_1(f)} = \frac{(L_1(f))^2}{\epsilon}$$

We define the function h to be,

$$h(\vec{x}) = \sum_{S \in \Lambda} \hat{f}(S) \chi_S(\vec{x}).$$

Using the Parseval identity we bound the error as follows,

$$E[(f-h)^2] = \sum_{S \notin \Lambda} \left(\hat{f}(S) - \hat{h}(S)\right)^2 = \sum_{S \notin \Lambda} \left(\hat{f}(S)\right)^2 \le \overbrace{\max_{S \notin \Lambda} |\hat{f}(S)|}^{\le \frac{\epsilon}{L_1(f)}} \cdot \overbrace{\sum_{S \notin \Lambda} |\hat{f}(S)|}^{\le L_1(f)} \le \epsilon.$$

The function h has $|\Lambda|$ non-zero coefficients, and the theorem follows from the bound on the size of Λ. ■

The above theorem defines a wide class of functions that can be learned efficiently using the Sparse Algorithm. Namely, if $L_1(f)$ is bounded by a polynomial then f can be learned in polynomial time by the Sparse Algorithm. In the next section we show that decision trees belong to this class of functions.

The following two simple claims would be helpful in bounding the L_1 norm of functions.

Claim 14.14 $L_1(f) + L_1(g) \ge L_1(f+g)$

Claim 14.15 $L_1(f) \cdot L_1(g) \ge L_1(f \cdot g)$

Sparse approximation of Decision Trees

Our aim is to show that decision trees have a small $L_1(f)$. This implies that decision trees can be approximated by a sparse function, and hence are learnable by the Sparse Algorithm. The following function would be helpful in describing decision trees.

Definition 14.16 *For $d \geq 1$ and a boolean vector $\vec{b} \in \{0,1\}^d$ we define the function $AND_{(b_1,\ldots,b_d)} : \{0,1\}^d \to \{0,1\}$ to be,*

$$AND_{(b_1,\ldots,b_d)}(\vec{x}) = \begin{cases} 1 & \forall i,\ 1 \leq i \leq d \quad b_i = x_i \\ 0 & Otherwise \end{cases}$$

Lemma 14.17 *For $d \geq 1$ and $(b_1,\ldots,b_d) \in \{0,1\}^d$,*

$$L_1(AND_{(b_1,\ldots,b_d)}) = 1$$

Proof: Rewrite the function $AND_{(b_1,\ldots,b_d)}$ as,

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

Using Claim 14.15,

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

Since $AND_{(b_1,\ldots,b_d)}(b_1,\ldots,b_d) = 1$, then the sum of the coefficients is at least 1, hence $L_1(AND) = 1$. ■

The following theorem bounds the L_1 norm of decision trees as a function of the number of nodes in the tree.

Theorem 14.18 *Let $f(\vec{x})$ be a function represented by a decision tree T. If T has m leaves, then,*

$$L_1(f) \leq m.$$

Proof: Given a leaf v of T, we look at path from the root of the tree to v. There are d_v nodes in this path. Let $x_{i_1},\ldots,x_{i_{d_v}}$ be the variables on this path. There is a unique assignment of values to $x_{i_1},\ldots,x_{i_{d_v}}$ such that f reaches the leaf v. Let $b_1^v,\ldots,b_{d_v}^v$ be this assignment. Then,

$$\forall \vec{x} \in \{0,1\}^n \quad AND_{(b_1^v,\ldots,b_{d_v}^v)}(x_{i_1},\ldots,x_{i_{d_v}}) = 1 \quad \Leftrightarrow \quad f(\vec{x}) \ \textit{arrive to the leaf } v.$$

The $AND_{(b^v_1,\ldots,b^v_{d_v})}$ function identifies all the inputs arriving to the leaf v. For every $v \in T$ let σ_v be the value returned by the leaf v. Since on any input f reaches exactly one leaf, it follows that,

$$\forall \vec{x} \in \{0,1\}^n \quad f(\vec{x}) = \sum_{v \in Leaves} \sigma_v \cdot AND_{(b^v_1,\ldots,b^v_{d_v})}(x_{i^v_1},\ldots,x_{i^v_{d_v}}).$$

Finally, by Claim 14.14 and Lemma 14.17,

$$L_1(f) \leq \sum_{v \in T} L_1(AND) \leq m \cdot L_1(AND) = m,$$

which completes the proof of the theorem. ■