Electronic Colloquium on Computational Complexity, Report No. 98 (2001)

On Learning Correlated Boolean Functions Using Statistical Query

Ke Yang
Computer Science Department,
Carnegie Mellon University,
5000 Forbes Ave., Pittsburgh, PA 15213, USA.
yangke@cs.cmu.edu [*]

November 19, 2001

Abstract

In this paper, we study the problem of using statistical query (SQ) to learn highly correlated boolean functions, namely, a class of functions where any pair agree on significantly more than a fraction 1/2 of the inputs. We give a limit on how well one can approximate all the functions without making any query, and then we show that beyond this limit, the number of statistical queries the algorithm has to make increases with the "extra" advantage the algorithm gains in learning the functions. Here the advantage is defined to be the probability the algorithm agrees with the target function minus the probability the algorithm doesn't agree.

An interesting consequence of our results is that the class of booleanized linear functions over a finite field ($f(_{\vec{a}}(\vec{x}) = 1$ iff $\phi(\vec{a}\cdot\vec{x}) = 1$, where $\phi : GF_p \mapsto \{-1, 1\}$ is an arbitrary boolean function the maps any elements in GF_p to ± 1). This result is useful since the hardness of learning booleanized linear functions over a finite field is related to the security of certain cryptosystems ([B01]). In particular, we prove that the class of linear threshold functions over a finite field ($f(_{\vec{a},b}(\vec{x}) = 1$ iff $\vec{a}\cdot\vec{x} \geq b$) cannot be learned efficiently using statistical query. This contrasts with Blum et al.'s result [BFK+96] that linear threshold functions over reals (perceptrons) are learnable using SQ model.

Finally, we describe a PAC-learning algorithm that learns a class of linear threshold functions in time that is provably impossible for statistical query algorithms to learn the class. With properly chosen parameters, this class of linear threshold functions can become an example of PAC-learnable, but not SQ-learnable functions that are not parity functions.

[*]An extended abstract to appear in the Proceedings of The Twelfth International Conference on Algorithmic Learning Theory (ALT'01), LNAI 2225.

ISSN 1433-8092

1 Introduction

Pioneered by Valiant [V84], machine learning theory is concerned with problems like "What class of functions can be efficiently learned under this learning model?". Among different learning models there are the Probably Approximately Correct model (PAC) by Valiant [V84] and the Statistical Query model (SQ) by Kearns [K98].

The SQ model is a restriction to the PAC model, where the learning algorithm doesn't see the samples with their labels, but only get the probabilities that a predicate is true: to be more precise, the learning algorithm provides a predicate $g(x, y)$ and a tolerance ϵ, and an SQ oracle returns a real number v that is ϵ-close to the expected value of $g(x, f(x))$ according a distribution of x, where f is the target functions. While seemingly a lot weaker than the PAC model, SQ model turns out to be very useful: in fact, a lot of known PAC learning algorithms are actually SQ model algorithms, or can be converted to SQ model algorithms. The readers are referred to [K98] for more comprehensive description.

One interesting feature for SQ model is that there are information-theoretical lower-bounds on the learnability of certain classes of functions. Kearns [K98] proved that parity functions cannot be efficiently learned in the SQ model. Blum et al. [BFJ+94] extended his result by showing that if a class of functions has "SQ-dimension" (informally, the maximum number of "almost un-correlated" functions in the class, where the correlation between two functions is the probability these two functions agree minus the probability they disagree) d, then a SQ learning algorithm has to make $\Omega(d^{1/3})$ queries, each of tolerance $O(d^{-1/3})$ in order to weakly learn $\mathcal{F}$. In [J00], Jackson further strengthened this lower bound by proving that $\Omega(2^n)$ queries are needed for an SQ-based algorithm to learn the class of parity functions over n bits. This result can be extended to any class of completely uncorrelated functions: $\Omega(d)$ queries are needed for an SQ-based algorithm to learn a class of functions if this class contains d functions that are completely "uncorrelated". Notice that this upper bound is optimal: [BFJ+94] proved that there are weak-learning algorithms for the class of functions using $O(d)$ queries.

In this paper, we study the problem of learning *correlated* functions. Suppose there is a class of boolean functions $\mathcal{F} = \{f_1, f_2, ..., f_d\}$, where any pair of functions f_i, f_j are highly correlated, namely f_i and f_j agree on a fraction $(1 + \lambda)/2$ of the inputs, where λ can be significantly larger than 0 (say, $\lambda = 1/3$). There are natural classes of correlated functions. An example is the "booleanized linear functions" in a finite field GF_p defined in this paper. Informally, these functions are of the form $f_{\vec{a}}(\vec{x}) = \phi(\vec{a}\cdot\vec{x})$, where ϕ (called a "booleanizer" function) is an arbitrary function that maps an element in GF_p to a boolean value ($+1$ or -1), and both $\vec{a}$ and $\vec{x}$ are vectors over GF_p. Booleanized linear functions can be viewed as natural extensions to parity functions (which are linear functions in GF_2), and intuitively, should be hard to learn by statistical query (since parity functions cannot be efficiently learned by statistical query). Actually they are (implicitly) conjectured to be hard to learn in general, and there are cryptosystems whose security is based on the assumption that booleanized linear functions are hard to learn. One example is the "blind encryption scheme" proposed by Baird [B01]: Roughly speaking, this private-key crypto-scheme picks a random $f_{\vec{a}}$ as the secret key, and encrypts a '0' bit by a random $\vec{x}$ such that $f_{\vec{a}}(\vec{x}) = 1$, and a '1' bit by a random $\vec{x}$ such that $f_{\vec{a}}(\vec{x}) = -1$ Knowing the secret key, decryption is just an invocation of the $f_{\vec{a}}$, which can be done very efficiently. Furthermore, it is (implicitly in [B01]) conjectured, that, by only inspecting random plaintext-ciphertext pairs $\langle \vec{x}, f_{\vec{a}}(\vec{x})\rangle$, it is hard to learn the function $f_{\vec{a}}$[1]. However, the results from [K98, BFJ+94, J00] don't immediately apply here since these booleanized linear functions are indeed correlated, and the correlation can be very large (in particular, [BFJ+94] requires the the correlation between any two functions to be $O(1/d^3)$, where for the booleanized linear functions, the correlation is of order $\Omega(1/d)$, and can even be constants).

Notice that in the case of correlated functions, the notion of "weak learning" can become trivial: if any pair of functions f_i and f_j have correlation λ, i.e., they agree on $(1 + \lambda)/2$ fraction of the inputs, then by always outputing $f_1(x)$ on every input x, an algorithm can approximate any function f_i with advantage at least λ, (the advantage of an algorithm is defined as the probability the algorithm predicts a function correctly minus the probability the algorithm predicts incorrectly). So if λ is non-negligibly larger than 0, this algorithm "weakly learns" the function class without even making *any* query to the target function.

In the first part of this paper, we prove that if there are d target functions $f_1, f_2, ..., f_d$, such that any pair f_i and f_j have almost the same correlation λ, then an algorithm can have maximally $\sqrt{\frac{1+(d-1)\lambda}{d}}$ advantage in approximating all the target functions, if no query is performed. Furthermore, we prove that in order to have any "extra" advantage S in learning a target function, about $\sqrt{d}\cdot S/2$ queries are needed. The result shows an advantage-query complexity trade-off: the more advantage one wants, the more queries one has to make. One consequence of our result is that booleanized linear functions cannot be learned efficiently using statistical query, and if the booleanizer is "almost biased" and the finite field GF_p is large, one cannot even weakly learn this class of functions. Our result provides some positive evidence towards the security of the blind encryption scheme by Baird.

[1] This is not exactly what the "blind encryption scheme" does, but is similar.

The technique we used in the proof, which could be of interest by itself, is to keep track of the "all-pair statistical distance" between scenarios when the algorithm is given different target functions — we denote this quantity by Δ We. prove that:

1. Before the algorithm makes queries, $\Delta = 0$.
2. After the algorithm finishes all the queries, Δ is "large".
3. Each query only increases Δ by a "small" amount.

And then we conclude that a lot of queries are needed in order to learn $\mathcal{F}$ well.

One interesting consequence from our result is that the class of linear threshold functions are not efficiently learnable. A linear threshold function in a finite field is defined as $f_{\vec{a},b}(\vec{x}) = 1$ if $\vec{a} \cdot \vec{x} \geq b$, and -1 otherwise, where $\vec{a} \in GF_p^n$ and $b \in GF_p$. These linear threshold functions over GF_p are interesting, since their counterparts over reals are well-known as "perceptrons" and they learnability are well studied. Blum et al. [BFK+96] proved that there are statistical query algorithms that learn linear threshold function over reals in polynomial time, even in the presence of noise. It is interesting to see this stark contrast.

In the second part of this paper, We present a learning algorithm, BUILD-TREE , that learns a class of linear threshold functions over finite fields where the threshold b is fixed to be $(p+1)/2$. Our algorithm uses a random example oracle, which produces a random pair $\langle \vec{x}, f_{\vec{a}}(\vec{x}) \rangle$ upon invocation. The algorithm's running time is $p^{O(n/\log n)}$ while the brute-force search algorithm takes time $p^{\Omega(n)}$, and any statistical query learning algorithm also has to take time $p^{\Omega(n)}$ to even weakly learn the functions. If we "pad" the input properly, we can make BUILD-TREE 's running time polynomial in the input size, while still no SQ learning algorithms can learn the class efficiently. This gives an example of PAC-learnable, but not SQ-learnable class of functions. Previously, both [K98] and [BFJ+94] proved that the class of parity functions fits into this category, and later [BKW00] proved that a class of noisy parity functions also fits. Our example is the first class of functions in this category that are not parity functions. This result provides some insights towards better understanding of SQ-learning algorithms.

The rest of the paper is organizes as follow: section 2 gives some notations and definitions to be used in this paper; section 3 proves a lower bound for SQ-learning algorithms; section 4 discusses the algorithm BUILD-TREE and its analysis.

Due to space constraint, the proofs to most lemmas and theorems are moved into the appendix.

2 Notations and Definitions

We give the notations and definitions to be used in the paper.

2.1 Functions and Oracles

Throughout this paper we are interested in functions whose input domain is a finite set Ω, where $|\Omega| = M$, and whose outputs domain is $\{-1, +1\}$. An input x to a function f is called a *positive example* if $f(x) = +1$, and *negative example* is $f(x) = -1$. Sometimes when the function f is clear from the context, we call the value of $f(x)$ the *label* of x. In a lot of cases, Ω takes a special form: $\Omega = GF_p^n$, where p is a prime number and n is a positive integer. In this case, we write an input in the vector form: $\vec{x}$, we use x^i to denote its i-th entry, an element in GF_p.

We now define the notion of learning a function. The overall model is an algorithm A with oracle access to a function f that A tries to learn (we call f the *target function*). A is given an input X and makes queries to the oracle. Finally A outputs a bit as its prediction of $A(X)$.

We use an "honest SQ-oracle" model, which is similar to the definition of "SQ-based algorithm" in [J00]:

Definition 1 *An* honest SQ-oracle *for function f takes two parameters g and N as inputs, where $g : GF_p^n \times \{-1, +1\} \to \{-1, +1\}$ is a function that takes an input in GF_p^n and a boolean input and outputs a binary value, and M is a positive integer written in* unary, *called the "sample count". The oracle returns $\frac{1}{N} \sum_{i=1}^{N} g(\vec{x}_i, f(\vec{x}_i))$ where each x_i is a random variable independently chosen according to a pre-determined distribution D. We denote this oracle by HSQ_f*

Notice that this definition of an honest SQ-oracle is different from the mostly-used definition of a "normal" SQ-oracle (sometimes denoted as $STAT_f$) as in [AD98, BFJ+94, BFK+96, BKW00, K98]. Actually an honest SQ-oracle is stronger than a "normal" SQ-oracle. Kearns [K98] proved that one can simulate a $STAT_f$ oracle efficiently in the PAC learning framework, and Decatur [D95] extensively studied the problem of efficiently simulating a $STAT_f$ oracle. Both their results can be easily extended to show that an honest SQ-oracle can be used to efficiently simulate a "normal" SQ-oracle. Therefore a lower bound with respect to an honest SQ-oracle automatically translates to a lower bound with respect to a "normal" SQ-oracle.

2.2 Bias and Inner Products of Functions

We define the *bias* of a real function f over Ω to be the *expected value* of f under a distribution D, and we denote that by $\langle f \rangle_D$:

$$\langle f \rangle_D = E_D[f(\vec{x})] = \sum_{x\in\Omega} D(x) f(x)$$

We define the *inner product* of two real functions over Ω to be the *expected value of* $f \cdot g$, denoted by $\langle f, g \rangle_D$:

$$\langle f, g \rangle_D = E_D[f(x)g(x)] = \sum_{x\in\Omega} D(x) \cdot f(x)g(x)$$

In the rest of the paper, we often omit the letter D if the distribution is clear from the context.

We can also view the inner product as the "correlation" between f and g. It is easy to verify that the definition of inner product is a proper one. Also it is important to observe that if f is a boolean functions, i.e., $f(x) \in \{-1, +1\}$, then $\langle f, f \rangle = 1$.

2.3 Approximating and Learning Functions

Given a function $f : \Omega \to \{-1, +1\}$ and an algorithm A which maps elements in Ω to -1 or $+1$, we can measure how well A approximates f. The algorithm could be a randomized one and thus the output of A on any input is a random variable. We define the *characteristic function* of algorithm A to be a real-valued function over the same domain Ω: $\psi_A : \Omega \to [-1, +1]$, such that

$$\psi_A(x) = 2 \cdot \mathsf{Pr}\,[A \text{ outputs } 1 \text{ on } x] - 1$$

where the probability is taken over the randomness A uses and, if A make oracle queries, the randomness from the oracles.

It is easy to verify that $\phi_A(x)$ is always within the range $[-1, 1]$. Given a probabilistic distribution D over Ω, we define the *advantage* of algorithm A in approximating function f to be

$$\langle f, \psi_A \rangle = \mathsf{Pr}_{\,A,D}[A \text{ agrees with } f \text{ on input } x] - \mathsf{Pr}_{\,A,D}[A \text{ disagrees with } f \text{ on input } x]$$

where the probability is taken over the randomness from A and the x that is randomly chosen from Ω according to D.

It is not hard to see that if A always agrees with f, then $\phi_A = f$, and the advantage of A in approximating f is 1; if A randomly guesses a value for each input, then $\phi_A \equiv 0$, and the advantage of A is 0.

For a class of functions $\mathcal{F}$, and an oracle algorithm A, we say A *approximates* $\mathcal{F}$ *with advantage* α if for every function $f \in \mathcal{F}$, the advantage of A in approximating f is at least α. In the case A queries an honest SQ-oracle HSQ_f in order to approximate the target function f, we say A *learns* $\mathcal{F}$ with advantage α with respect to an honest SQ-oracle.

We note that the "advantage" measure for learning a function isn't very different from the more commonly used "accuracy/confidence" measure in PAC learning. Recall that an algorithm learns $\mathcal{F}$ with accuracy ϵ and confidence δ, if for any $f \in \mathcal{F}$, the algorithm A, using an oracle about f, with probability at least $1 - \delta$, agrees with f with probability at least $1 - \epsilon$. It is easy to prove the following facts:

Lemma 1 *Let $\mathcal{F}$ be a class of boolean functions over Ω, and let A be an oracle algorithm. If A learns $\mathcal{F}$ with accuracy ϵ and confidence δ, then A learns $\mathcal{F}$ with advantage at least $1 - 2\epsilon - 2\delta$. On the other hand, if A learns $\mathcal{F}$ with advantage at least α, then A learns $\mathcal{F}$ with accuracy ϵ and confidence δ for any (ϵ, δ) pair satisfying*

$$\alpha \geq 1 - 2\epsilon\delta$$

Proof: First, if A learns $\mathcal{F}$ with accuracy ϵ and confidence δ, then we know with probability $1 - \delta$, the advantage of A is at least $(1 - \epsilon) - \epsilon = 1 - 2\epsilon$, and with probability δ, the advantage of A is at least -1. So the overall advantage of A is at least

$$(1 - \delta)(1 - 2\epsilon) - \delta \geq 1 - 2\epsilon - 2\delta$$

Second, if A learns $\mathcal{F}$ with advantage α, we translate that into the PAC language. We define A to be "lucky", if its advantage is at least $1 - 2\epsilon$. We denote the probability that A is lucky by p, and then we have

$$\alpha \leq p + (1 - p)(1 - 2\epsilon)$$

or

$$\alpha \leq 1 - 2\epsilon + 2\epsilon p$$

So if we have

$$\alpha \geq 1 - 2\epsilon\delta$$

then we have $\delta > 1 - p$ and A is lucky with probability at least $1 - \delta$. Therefore A learns $\mathcal{F}$ with accuracy ϵ and confidence δ. ■

Therefore, roughly speaking: if an algorithm A learns $\mathcal{F}$ with high confidence and high accuracy (A "strongly" learns $\mathcal{F}$), then the advantage of A in learning $\mathcal{F}$ is close to 1; if A learns $\mathcal{F}$ weakly, then the advantage of A is non-negligibly higher than 0. On the other hand, if the advantage A has in learning $\mathcal{F}$ is close to 1, then A (strongly) learns $\mathcal{F}$.

The reason that we use the advantage measure in this paper is that we want to show a continuous "trade-off" result between how many queries are needed and how "well" an algorithm learns $\mathcal{F}$, and using one single parameter makes the discussion more convenient.

2.4 Booleanized Linear Functions and Linear Threshold Functions in Finite Fields

Suppose p is a prime number and n a positive integer. Given an arbitrary function that maps inputs from GF_p to boolean values,

$$\phi : GF_p \to \{-1, +1\}$$

we define a class $\mathcal{F}_\phi$ of *booleanized linear functions* as a collections of boolean functions:

$$\mathcal{F}_\phi = \{f_{\vec{a},\phi}(\vec{x}) := \phi(\vec{a} \cdot \vec{x}) \mid \vec{a} \in GF_p^n\},$$

and we call function ϕ the *booleanizer.*

Booleanized linear functions can be viewed as natural extensions of parity functions (which are linear functions over GF_2^n).

If the booleanizer function, ϕ, is a threshold function:

$$\phi_b(x) = \begin{cases} 1 & , \quad \text{if } x \geq b \\ -1 & , \quad \text{if } x < b \end{cases}$$

we call the corresponding class of booleanized linear functions *linear threshold functions*, and denote the functions by $f_{\vec{a},b}$. Here the comparison is performed by first mapping elements in GF_p to integers $\{0, 1, ..., p-1\}$ in the most straightforward way and then performing integer comparison.

2.5 The Tensor Product and Statistical Distance

Given two probabilistic distributions D and D' over spaces Λ and Λ', we define their *tensor product* $D \otimes D'$ to be a new distribution over $\Lambda \times \Lambda'$:

$$\mathsf{Pr}_{\,D\otimes D'}[(X, X') = (x, x')] = \mathsf{Pr}_{\,D}[X = x] \cdot \mathsf{Pr}_{\,D'}[X' = x']$$

Given a finite space Λ and distributions $D_1, D_2, ..., D_m$ over Λ, we define the *all-pair* L_2 *statistical distance* (abbreviated as SD) among $D_1, D_2, ..., D_m$ to be

$$\mathsf{SD}_2(D_1, D_2, ..., D_m) = \left[\sum_{i=1}^{m}\sum_{j=1}^{m}\sum_{x\in\Lambda}\left(\mathsf{Pr}_{\,D_i}[X = x] - \mathsf{Pr}_{\,D_j}[X = x]\right)^2\right]^{\frac{1}{2}}$$

Under this definition, it is easy to see that

$$\mathsf{SD}_2(D, D) = 0$$

and

$$\mathsf{SD}_2(D_1, D_2, ..., D_m) = \left[\sum_{i=1}^{m}\sum_{j=1}^{m}\mathsf{SD}_2(D_i, D_j)^2\right]^{\frac{1}{2}}$$

One useful property of the all-pair L_2 statistical distance is the sub-additivity:

Lemma 2 *Let $D_1, D_2, ..., D_m$ be distributions over Λ and $D'_1, D'_2, ..., D'_m$ be distributions over Λ'. Then we have*

$$\mathsf{SD}_2(D_1 \otimes D'_1, D_2 \otimes D'_2, ..., D_m \otimes D'_m) \leq \mathsf{SD}_2(D_1, D_2, ..., D_m) + \mathsf{SD}_2(D'_1, D'_2, ..., D'_m)$$

Proof: We define $P_{i,x} = \mathsf{Pr}_{\ D_i}[X = x]$ and $Q_{i,y} = \mathsf{Pr}_{\ D'_i}[Y = y]$. Then we have

$$\mathsf{Pr}_{\ D_i \otimes D'_i}[(X,Y) = (x,y)] = \mathsf{Pr}_{\ D_i}[X = x] \cdot \mathsf{Pr}_{\ D'_i}[Y = y] = P_{i,x} Q_{i,y}$$

We denote $\mathsf{SD}_2(D_1 \otimes D'_1, D_2 \otimes D'_2, ..., D_m \otimes D'_m)$ by D. Then we have

$$\begin{aligned}
D &= \left[\sum_{i=1}^{m}\sum_{j=1}^{m}\sum_{x}\sum_{y}(P_{i,x}Q_{i,y} - P_{j,x}Q_{j,y})^2\right]^{\frac{1}{2}} \\
&= \left[\sum_{i=1}^{m}\sum_{j=1}^{m}\sum_{x}\sum_{y}(P_{i,x}Q_{i,y} - P_{i,x}Q_{j,y} + P_{i,x}Q_{j,y} - P_{j,x}Q_{j,y})^2\right]^{\frac{1}{2}} \\
&\leq \left[\sum_{i=1}^{m}\sum_{j=1}^{m}\sum_{x}\sum_{y}(P_{i,x}Q_{i,y} - P_{i,x}Q_{j,y})^2\right]^{\frac{1}{2}} + \left[\sum_{i=1}^{m}\sum_{j=1}^{m}\sum_{x}\sum_{y}(P_{i,x}Q_{j,y} - P_{j,x}Q_{j,y})^2\right]^{\frac{1}{2}} \\
&= \left[\sum_{i=1}^{m}\sum_{j=1}^{m}\sum_{y}(Q_{i,y} - Q_{j,y})^2 \cdot \sum_{x} P_{i,x}^2\right]^{\frac{1}{2}} + \left[\sum_{i=1}^{m}\sum_{j=1}^{m}\sum_{x}(P_{i,x} - P_{j,x})^2 \cdot \sum_{y} Q_{j,x}^2\right]^{\frac{1}{2}} \\
&\leq \left[\sum_{i=1}^{m}\sum_{j=1}^{m}\sum_{y}(Q_{i,y} - Q_{j,y})^2\right]^{\frac{1}{2}} + \left[\sum_{i=1}^{m}\sum_{j=1}^{m}\sum_{x}(P_{i,x} - P_{j,x})^2\right]^{\frac{1}{2}} \\
&= \mathsf{SD}_2(D_1, D_2, ..., D_m) + \mathsf{SD}_2(D'_1, D'_2, ..., D'_m)
\end{aligned}$$

where the first inequality is by the triangle inequality and the second inequality is because

$$\sum_x P_{i,x}^2 \leq \left(\sum_x P_{i,x}\right)^2 = 1$$

■

Since each random variable naturally induces a distribution, we can also define all-pair L_2 statistical distance among random variables: For random variables $X_1, X_2, ..., X_m$, their all-pair L_2 statistical distance is defined to be the all-pair L_2 statistical distance among the the distributions induced by them. The sub-additivity property remains true: suppose we have random variables $X_1, X_2, ..., X_m$ and $Y_1, Y_2, ..., Y_m$, such that X_i is independent to Y_j for any $i, j \in \{1, 2, ..., m\}$, we have

$$\mathsf{SD}_2(X_1Y_1, X_2Y_2, ..., X_mY_m) \leq \mathsf{SD}_2(X_1, X_2, ..., X_m) + \mathsf{SD}_2(Y_1, Y_2, ..., Y_m)$$

2.6 Chernoff Bounds

We will be using Chernoff bounds in our paper, and our version if from [MR95].

Theorem 1 *Let $X_1, X_2, ..., X_n$ be a sequence of n independent $\{0,1\}$ random variables. Let S be the sum of the random variables and $\mu = E[S]$. Then, for $0 \leq \delta \leq 1$, the following inequalities hold:*

$$\mathsf{Pr}\,[S > (1+\delta)\mu] \leq e^{-\delta^2\mu/3}$$

and

$$\mathsf{Pr}\,[S < (1-\delta)\mu] \leq e^{-\delta^2\mu/2}$$

■

3 Statistical Query Model: Negative Results

In this section we present a negative result characterizing the Statistical Query Model.

Throughout this section, we use Ω to denote a finite set of size M and we are interested in functions take inputs from Ω and output $+1$ or -1.

3.1 Statistical Dimension and Fourier Analysis

Definition 2 *Let Ω be a finite set and let $\mathcal{F}$ be a class of boolean functions whose input domain is Ω, and D a distribution over Ω, we define* SQ-DIM$(\mathcal{F}, D)$*, the* statistical query dimension *of $\mathcal{F}$ with respect to D, to be the largest natural number d such that there exists a real number λ, satisfying $0 \le \lambda \le 1/2$ and $\mathcal{F}$ contains d functions $f_1, f_2, ..., f_d$ with the property that for all $i \neq j$, we have*

$$|\langle f_i, f_j\rangle - \lambda| \le \frac{1}{d^3}$$

Notice the definition of SQ-DIM in [BFJ+94] can be regarded as the special case where $\Omega = \{-1, +1\}^n$ with the restriction that $\lambda = 0$.

Notice though each of the functions $f_1, f_2, ..., f_d$ can be highly correlated to others, since the correlation is always almost the same, we call it a "false correlation". As we will prove in the next lemma, we can "extract" d new functions $\tilde{f}_1, \tilde{f}_2, ..., \tilde{f}_d$ from $f_1, f_2, ..., f_d$, such that the new functions are almost totally uncorrelated to each other.

Lemma 3 *Let Ω, D, d, λ, and $f_1, f_2, ..., f_d$ be as defined in definition 2, and $\lambda > 0$. We define d real-valued functions $\tilde{f}_1, \tilde{f}_2, ..., \tilde{f}_d$:*

$$\tilde{f}_i(x) = \frac{1}{\sqrt{1-\lambda}} f_i(x) - \frac{1}{d} \cdot \left(\frac{1}{\sqrt{1-\lambda}} - \frac{1}{\sqrt{1+(d-1)\lambda}} \right) \cdot \sum_{j=1}^{d} f_j(x) \tag{1}$$

Then we have

$$|\langle \tilde{f}_i, \tilde{f}_i \rangle - 1| \le \frac{8}{d^3}, \ \forall i \tag{2}$$

and

$$|\langle \tilde{f}_i, \tilde{f}_j \rangle| \le \frac{8}{d^3}, \ \forall i \neq j \tag{3}$$

Proof: One could directly substitute the definition of $\tilde{f}_1, ..., \tilde{f}_d$ into the formula and check, but here is the reasoning:

We first define a new function $\bar{f}$ which is the average of functions $f_1, f_2, ..., f_d$:

$$\bar{f}(x) = \frac{1}{d} \sum_{i=1}^{d} f_i(x)$$

Then we can work out the inner product of $\bar{f}$ and f_i:

$$\langle \bar{f}, f_i \rangle = \frac{1}{d} \sum_{j=1}^{d} \langle f_i, f_j \rangle = \frac{1}{d} + \frac{1}{d} \sum_{j \neq i} \langle f_i, f_j \rangle$$

So if we define $\alpha = (\frac{1}{d} + \frac{d-1}{d}\lambda)$, we have

$$|\langle \bar{f}, f_i \rangle - \alpha)| \le \frac{1}{d^3}$$

and since $\bar{f}(x) = \frac{1}{d}\sum_{i=1}^{d} f_i(x)$, we also have

$$|\langle \bar{f}, \bar{f} \rangle - \alpha)| \le \frac{1}{d^3}$$

Therefore we know that $\alpha \ge 0$ since $\langle \bar{f}, \bar{f} \rangle \ge 0$.

We define $g_i(x) = f_i(x) - Z\bar{f}(x)$ for $i = 1, 2, ..., d$. where Z is a constant to be decided. Now we compute the inner products of g_i and g_j:

$$\begin{aligned} \langle g_i, g_j \rangle &= \langle f_i - Z\bar{f}, f_j - Z\bar{f} \rangle \\ &= \langle f_i, f_j \rangle - Z\langle f_i, \bar{f} \rangle - Z\langle f_i, \bar{f} \rangle + Z^2 \langle \bar{f}, \bar{f} \rangle \\ &= \lambda - 2\alpha Z + \alpha Z^2 + O(\frac{1}{d^3}) \end{aligned}$$

Let

$$Z = 1 - \sqrt{\frac{\alpha - \lambda}{\alpha}} = 1 - \sqrt{\frac{1-\lambda}{1+(d-1)\lambda}}$$

Then Z is a solution to the equation

$$\lambda - 2\alpha Z + \alpha Z^2 = 0$$

and $0 \le Z < 1$. So we can show that

$$|\langle g_i, g_j\rangle| \le \frac{4}{d^3}$$

Now if we compute the inner product of g_i with itself, we get

$$\begin{aligned}\langle g_i, g_i\rangle &= \langle f_i - Z\bar{f}, f_i - Z\bar{f}\rangle \\ &= \langle f_i, f_i\rangle - Z\langle f_i, \bar{f}\rangle - Z\langle f_i, \bar{f}\rangle + Z^2\langle \bar{f}, \bar{f}\rangle \\ &= 1 - 2\alpha Z + \alpha Z^2 + O(\frac{1}{d^3})\end{aligned}$$

So we have

$$|\langle g_i, g_i\rangle - (1-\lambda)| \le \frac{4}{d^3}$$

Finally we define

$$\tilde{f}_i(x) = \frac{1}{\sqrt{1-\lambda}} g_i(x) = \frac{1}{\sqrt{1-\lambda}} f_i(x) - \left(\frac{1}{\sqrt{1-\lambda}} - \frac{1}{\sqrt{1+(d-1)\lambda}}\right)\bar{f}(x) \tag{4}$$

and we have

$$|\langle \tilde{f}_i, \tilde{f}_i\rangle - 1| \le \frac{8}{d^3}, \ \forall i$$

and

$$|\langle \tilde{f}_i, \tilde{f}_j\rangle| \le \frac{8}{d^3}, \ \forall i \ne j$$

because we have $0 \le \lambda \le 1/2$ and therefore $\frac{1}{1-\lambda} \le 2$. ■

So we now get a group of functions $\tilde{f}_1, \tilde{f}_2, ..., \tilde{f}_d$ that are "almost" orthogonal.

Next, we can extend this group of functions to a basis and perform Fourier analysis on the basis. The part of analysis are very similar to the proofs in [BFJ+94], but with different parameters and (sometimes) improved bounds.

Lemma 4 *Let Ω, D, d, λ, and $f_1, f_2, ..., f_d$ be as defined in definition 2, and suppose that D has full support, i.e., $\forall x \in \Omega, D(x) > 0$, and $\lambda > 0$, $d > 16$. Let functions $\tilde{f}_1, \tilde{f}_2, ..., \tilde{f}_d$ be as defined in lemma 3, then there exist functions $\tilde{f}_{d+1}, ...\tilde{f}_M$, such that $\{\tilde{f}_1, ..., \tilde{f}_M\}$ form a basis for the vector space of all real functions over Ω. Furthermore, the added functions are orthonormal, namely, for any $j > d$ and any k, $\langle \tilde{f}_j, \tilde{f}_k\rangle = 0$ and $\langle \tilde{f}_j, \tilde{f}_j\rangle = 1$.*

Proof: We first prove that $\tilde{f}_1, \tilde{f}_2, ..., \tilde{f}_d$ are linear independent. Otherwise we assume that $\tilde{f}_1 = \sum_{j>1} \alpha_j \tilde{f}_j$ for some $\alpha_2, ..., \alpha_d$. Then we have

$$\begin{aligned}0 &= E_D\left[(\tilde{f}_1 - \sum_{j>1}\alpha_j \tilde{f}_j)^2\right] \\ &= E_D[\tilde{f}_1^{\,2}] - 2\sum_{j>1}\alpha_j E_D[\tilde{f}_1\tilde{f}_j] + \sum_{j,k>1}\alpha_j\alpha_k E_D[\tilde{f}_j\tilde{f}_k]\end{aligned}$$

We use define α_{max} to be

$$\alpha_{max} = \max\{|\alpha_j| \ : \ j > 1\}$$

and thus we have

$$\sum_{j>1} |\alpha_j|^2 \ge \alpha_{max}^2,$$

$$\left|\sum_{j>1}\alpha_j E_D[\tilde{f}_1\tilde{f}_j]\right| \le \frac{8\alpha_{max}}{d^2},$$

and

$$\left|\sum_{j,k>1, j\ne k}\alpha_j\alpha_k E_D[\tilde{f}_j\tilde{f}_k]\right| \le \frac{8\alpha_{max}^2}{d}$$

So we have

$$0 \le (1 - \frac{8}{d^3}) + \alpha_{max}^2 - \frac{16\alpha_{max}}{d^2} - \frac{8\alpha_{max}^2}{d}$$

If $\alpha_{max} \le 1$, we have

$$(1 - 8/d^3) + \alpha_{max}^2 - \frac{16\alpha_{max}}{d^2} - \frac{8\alpha_{max}^2}{d} \ge 1 - \frac{8}{d^3} - \frac{16}{d^2} - \frac{8}{d}$$

which is more than 0 when $d > 16$.

If $\alpha_{max} > 1$, we have

$$(1 - 8/d^3) + \alpha_{max}^2 - \frac{16\alpha_{max}}{d^2} - \frac{8\alpha_{max}^2}{d} \geq 1 - \frac{8}{d^3} + \alpha_{max}^2(1 - \frac{16}{d^2} - \frac{8}{d}) \geq 1 - \frac{8}{d^3} > 0$$

So either way we have a contradiction.

Now we already have d linear independent functions $\tilde{f}_1, ...\tilde{f}_d$, we can use Gram-Schmidt process to extend them to a basis $\tilde{f}_1, \tilde{f}_2, ..., \tilde{f}_d, \tilde{f}_{d+1}, ..., \tilde{f}_M$. for the vector space of all real functions over Ω, and make sure the added functions are orthonormal: for any $j > d$ and any k, $\langle \tilde{f}_j, \tilde{f}_k \rangle = 0$ and $\langle \tilde{f}_j, \tilde{f}_j \rangle = 1$. ■

Now that we have a basis for real functions over Ω, we can extend the distribution D to a distribution $\tilde{D}$ over $\Omega \times \{-1, +1\}$, and extend the basis to a basis for real functions over $\Omega \times \{-1, +1\}$.

Lemma 5 *Let Ω, D, d, λ, and functions $\tilde{f}_1, \tilde{f}_2, ..., \tilde{f}_M$ be as defined in lemma 4. We define a new distribution $\tilde{D}$ over $\Omega \times \{-1, +1\}$ as:*

$$\tilde{D}(x, -1) = \tilde{D}(x, +1) = D(x)/2$$

In other words, $\tilde{D}$ is the tensor product of distribution D over Ω and the uniform distribution over $\{-1, +1\}$. We extend the definitions of $\tilde{f}_1, ..., \tilde{f}_M$ to the input domain $\Omega \times \{-1, +1\}$ by defining

$$\tilde{f}_i(x, y) = \tilde{f}_i(x)$$

for $x \in \Omega$ and $y \in \{-1, +1\}$. We also define M new functions $h_1, h_2, ..., h_d$ over $\Omega \times \{-1, +1\}$:

$$h_i(x, y) = \tilde{f}_i(x) \cdot y$$

for $x \in \Omega$ and $y \in \{-1, +1\}$. Then $\{\tilde{f}_2, ..., \tilde{f}_M, h_1, ..., h_M\}$ form a basis for the real functions over $\Omega \times \{-1, +1\}$.

Proof: It is easy to check that

$$\begin{aligned} \langle \tilde{f}_i, h_j \rangle &= \frac{1}{2} E_D[\tilde{f}_i(x) \cdot h_j(x, -1)] + \frac{1}{2} E_D[\tilde{f}_i(x) \cdot h_j(x, +1)] \\ &= \frac{1}{2} E_D[-\tilde{f}_i(x) \cdot h_j(x)] + \frac{1}{2} E_D[\tilde{f}_i(x) \cdot h_j(x)] \\ &= 0 \end{aligned}$$

and

$$\begin{aligned} \langle h_i, h_j \rangle &= \frac{1}{2} E_D[h_i(x, -1) \cdot h_j(x, -1)] + \frac{1}{2} E_D[h_i(x, +1) \cdot h_j(x, +1)] \\ &= E_D[\tilde{f}_i(x), \tilde{f}_j(x)] \end{aligned}$$

■

Now that we have a basis for real functions over $\Omega \times \{-1, +1\}$, we can perform a Fourier Analysis for any function over $\Omega \times \{-1, +1\}$.

Definition 3 *Let Ω, D, d, and functions $\tilde{f}_1, \tilde{f}_2, ..., \tilde{f}_M$, h_1, ..., h_M be as defined in lemma 5. Let g be an arbitrary function over $\Omega \times \{-1, +1\}$, we can write g (uniquely) as*

$$g(x, y) = \sum_{i=1}^{M} \alpha_i \tilde{f}_i(x) + \sum_{i=1}^{M} \beta_i h(x, y) \tag{5}$$

We call $(\alpha_1, ..., \alpha_M, \beta_1, ..., \beta_M)$ the Fourier Coefficients *of function g.*

Notice the basis isn't an orthonormal one, but it is close. The following lemmas show the upper bounds for the coefficients.

Lemma 6 *Let Ω, D, d, and functions $\tilde{f}_1, \tilde{f}_2, ..., \tilde{f}_M$, h_1, ..., h_M be as defined in lemma 5. Let g be an arbitrary function over $\Omega \times \{-1, +1\}$, such that $|g(x, y)| \leq 1$ for all $x \in \Omega$ and $y \in \{-1, +1\}$. We write g in its Fourier coefficients as in definition 3. Then, for $i = 1, 2, ..., M$, $|\alpha_i| \leq 1 + 10/d^2$ and $|\beta_i| \leq 1 + 10/d^2$.*

Proof: WLOG we assume that $|\alpha_1|$ is the largest coefficient. We look at the projection of g on $\tilde{f}_1$:

$$\begin{aligned}\langle \tilde{f}_1, g\rangle \tilde{f}_1 &= \left(\alpha_1 + \sum_{i>1} \alpha_i \langle \tilde{f}_1, \tilde{f}_i\rangle + \sum_{i>1} \beta_i \langle \tilde{f}_1, h_i\rangle\right) \tilde{f}_1 \\ &= \left(\alpha_1 + \sum_{i=2}^{d} \alpha_i \langle \tilde{f}_1, \tilde{f}_i\rangle\right) \tilde{f}_1\end{aligned}$$

And since this is a projection, we know it is no greater than $||g||$, which bounded by 1. So we have

$$\begin{aligned}1 \geq ||g|| &\geq |\langle \tilde{f}_1, g\rangle \tilde{f}_1| \cdot |\langle \tilde{f}_1, \tilde{f}_1\rangle| \\ &\geq (|\alpha_1| - \sum_{i=2}^{d} |\alpha_i| \cdot \frac{8}{d^3}) \cdot (1 - \frac{8}{d^3}) \\ &\geq |\alpha_1|(1 - \frac{8}{d^2}) \cdot (1 - \frac{8}{d^3}) \\ &\geq |\alpha_1|/(1 + \frac{10}{d^2})\end{aligned}$$

when $d > 16$. ■

Lemma 7 *Let Ω, D, d, functions $\tilde{f}_1, \tilde{f}_2, ..., \tilde{f}_M$, h_1, ..., h_M and g be as defined in lemma 6, where $d > 100$. Then we have*

$$\sum_{i=1}^{M} \beta_i^2 \leq 1 + \frac{100}{d} \tag{6}$$

and

$$\sum_{i=1}^{M} \alpha_i^2 \leq 1 + \frac{100}{d} \tag{7}$$

Proof: we have

$$\begin{aligned}1 \geq ||g||^2 &= |\sum_{i=1}^{M} \alpha_i \tilde{f}_i + \sum_{i=1}^{M} \beta_i h_i|^2 \\ &\geq \sum_{i=1}^{M} |\alpha_i|^2 \langle \tilde{f}_i, \tilde{f}_i\rangle + \sum_{i=1}^{M} |\beta_i|^2 \langle h_i, h_i\rangle - \sum_{i\neq j} |\alpha_i\alpha_j \langle \tilde{f}_i, \tilde{f}_j\rangle| - \sum_{i\neq j} |\beta_i\beta_j \langle h_i, h_j\rangle| - \sum_{i,j} |\alpha_i\beta_j \langle \tilde{f}_i, h_j\rangle| \\ &\geq \sum_{i=1}^{d} |\beta_i|^2 \langle h_i, h_i\rangle - 2 \cdot \sum_{1\leq i<j\leq d} |\alpha_i\alpha_j \langle \tilde{f}_i, \tilde{f}_j\rangle| - 2 \cdot \sum_{1\leq i<j\leq d} |\beta_i\beta_j \langle h_i, h_j\rangle| \\ &> (1 - \frac{8}{d^3}) \sum_{i=1}^{d} |\beta_i|^2 - \frac{96}{d}\end{aligned}$$

So when $d > 100$, we have $\sum_{i=1}^{d} |\beta_i|^2 \leq (1 + \frac{96}{d})/(1 - \frac{8}{d^3}) \leq 1 + \frac{100}{d}$.

The proof for the other inequality is similar. ■

An immediate corollary from lemma 7 is:

Lemma 8 *Let Ω, D, d, functions $\tilde{f}_1, \tilde{f}_2, ..., \tilde{f}_M$, h_1, ..., h_M and g be as defined in lemma 6, where $d > 100$. Then we have*

$$\sum_{i=1}^{d} |\beta_i| \leq \sqrt{d} \cdot (1 + \frac{50}{d}) \tag{8}$$

and

$$\sum_{i=1}^{d} |\alpha_i| \leq \sqrt{d} \cdot (1 + \frac{50}{d}) \tag{9}$$

Proof: By Cauchy-Schwartz inequality and lemma 7,

$$\begin{aligned}\sum_{i=1}^{d} |\beta_i| &\leq \left(d \cdot \sum_{i=1}^{d} |\beta_i|^2\right)^{\frac{1}{2}} \\ &\leq \sqrt{d + 100} \\ &\leq \sqrt{d} \cdot (1 + \frac{50}{d})\end{aligned}$$

The proof for the other inequality is similar. ■

The next lemma gives a general bound for how the coefficients are related to the inner products.

Lemma 9 *Let Ω, D, d, and functions $\tilde{f}_1, \tilde{f}_2, ..., \tilde{f}_M$, be as defined in definition 4. Let ϕ be a real-valued function over Ω such that $|g(x)| \leq 1$ for all $x \in \Omega$. We write down g as*

$$g(x) = \sum_{i=1}^{M} \alpha_i \tilde{f}_i(x) \tag{10}$$

Then

$$\langle g, f_j \rangle \leq \alpha_j \cdot \sqrt{1-\lambda} + \left(1 - \sqrt{\frac{1-\lambda}{1+(d-1)\lambda}}\right) \sqrt{\frac{1+(d-1)\lambda}{d}} + \frac{60}{d} \tag{11}$$

Proof: By lemma 3, we have

$$f_i(x) = \sqrt{1-\lambda}\tilde{f}_i(x) + \left(1 - \sqrt{\frac{1-\lambda}{1+(d-1)\lambda}}\right) \bar{f}(x)$$

for $i = 1, 2, ..., d$. Summing the expression above for $i = 1, 2, ..., d$, we get

$$d\bar{f}(x) = \sum_{i=1}^{d} f_i(x) = \sqrt{1-\lambda} \sum_{i=1}^{d} \tilde{f}_i(x) + d\left(1 - \sqrt{\frac{1-\lambda}{1+(d-1)\lambda}}\right) \bar{f}(x)$$

or

$$\bar{f}(x) = \frac{\sqrt{1+(d-1)\lambda}}{d} \cdot \sum_{i=1}^{d} \tilde{f}_i(x)$$

Therefore

$$\begin{aligned} f_i(x) &= \sqrt{1-\lambda}\tilde{f}_i(x) + \left(1 - \sqrt{\frac{1-\lambda}{1+(d-1)\lambda}}\right) \bar{f}(x) \\ &= \sqrt{1-\lambda}\tilde{f}_i(x) + \left(1 - \sqrt{\frac{1-\lambda}{1+(d-1)\lambda}}\right) \frac{\sqrt{1+(d-1)\lambda}}{d} \cdot \sum_{i=1}^{d} \tilde{f}_i(x) \end{aligned}$$

We are only interested in the coefficients $\alpha_1, \alpha_2, ..., \alpha_d$, and thus we define

$$h(x) = \sum_{i=d+1}^{M} \alpha_i \tilde{f}_i(x)$$

Then we have $g(x) = h(x) + \sum_{i=1}^{d} \alpha_i \tilde{f}_i(x)$, and $\langle h, \tilde{f}_i \rangle = 0$ for $i = 1, 2, ..., d$. Now we compute the inner product of g and f_i:

$$\begin{aligned} \langle g, f_i \rangle &= \langle h(x) + \sum_{i=1}^{d} \alpha_i \tilde{f}_i(x),\ \sqrt{1-\lambda}\tilde{f}_i(x) + \left(1 - \sqrt{\frac{1-\lambda}{1+(d-1)\lambda}}\right) \frac{\sqrt{1+(d-1)\lambda}}{d} \cdot \sum_{i=1}^{d} \tilde{f}_i(x) \rangle \\ &= \alpha_i \sqrt{1-\lambda} \langle \tilde{f}_i, \tilde{f}_i \rangle + \sqrt{1-\lambda} \sum_{j \neq i} \alpha_j \langle \tilde{f}_i, \tilde{f}_j \rangle + \left(1 - \sqrt{\frac{1-\lambda}{1+(d-1)\lambda}}\right) \frac{\sqrt{1+(d-1)\lambda}}{d} \cdot \left(\sum_{i=1}^{d} \alpha_i \tilde{f}_i(x)\right) \cdot \left(\sum_{i=1}^{d} \tilde{f}_i(x)\right) \\ &\leq \alpha_i \cdot \sqrt{1-\lambda} + \sqrt{1-\lambda} \left(\sum_{i=1}^{d} \alpha_i \frac{8}{d^3}\right) + \left(1 - \sqrt{\frac{1-\lambda}{1+(d-1)\lambda}}\right) \frac{\sqrt{1+(d-1)\lambda}}{d} \cdot \left(\sum_{i=1}^{d} \alpha_i\right) (1 + d^2 \cdot \frac{8}{d^3}) \\ &= \alpha_i \cdot \sqrt{1-\lambda} + \left(1 - \sqrt{\frac{1-\lambda}{1+(d-1)\lambda}}\right) \frac{\sqrt{1+(d-1)\lambda}}{d} \cdot \left(\sum_{i=1}^{d} \alpha_i\right) + \Delta \end{aligned}$$

where

$$\begin{aligned} \Delta &= \sqrt{1-\lambda} \left(\sum_{i=1}^{d} \alpha_i \frac{8}{d^3}\right) + \left(1 - \sqrt{\frac{1-\lambda}{1+(d-1)\lambda}}\right) \frac{\sqrt{1+(d-1)\lambda}}{d} \cdot \left(\sum_{i=1}^{d} \alpha_i\right) (d^2 \cdot \frac{8}{d^3}) \\ &\leq \frac{9}{d} \left(\sum_{i=1}^{d} \alpha_i\right) \cdot \frac{\sqrt{1+(d-1)\lambda}}{d} \end{aligned}$$

$$\begin{aligned}
&\leq \frac{9}{d^{3/2}}\left(\sum_{i=1}^{d}\alpha_i\right)\\
&\leq \frac{10}{d}
\end{aligned}$$

since we have by lemma 8. Therefore,

$$\begin{aligned}
\langle g, f_i\rangle &\leq \alpha_i\cdot\sqrt{1-\lambda}+\left(1-\sqrt{\frac{1-\lambda}{1+(d-1)\lambda}}\right)\frac{\sqrt{1+(d-1)\lambda}}{d}\cdot\left(\sum_{i=1}^{d}\alpha_i\right)+\frac{10}{d}\\
&\leq \alpha_i\cdot\sqrt{1-\lambda}+\left(1-\sqrt{\frac{1-\lambda}{1+(d-1)\lambda}}\right)\frac{\sqrt{1+(d-1)\lambda}}{d}\cdot\left(\sqrt{d}\cdot(1+\frac{50}{d})\right)+\frac{10}{d}\\
&\leq \alpha_i\cdot\sqrt{1-\lambda}+\left(1-\sqrt{\frac{1-\lambda}{1+(d-1)\lambda}}\right)\frac{\sqrt{1+(d-1)\lambda}}{d}+\frac{60}{d}
\end{aligned}$$

■

3.2 Approximating a Function without a query

We give an upper bound on the advantage an algorithm A can have to approximating a class of functions, if A doesn't make any queries.

Theorem 2 *Let Ω be a domain of size M, let D be a probabilistic distribution over Ω. Let $\mathcal{F}$ be a class of functions $\mathcal{F}=\{f_1,f_2,...,f_d\}$, such that $|\langle f_i,f_j\rangle-\lambda|\leq 1/d^3$ for all pairs $i\neq j$, where $\lambda>0$. Let $g:\Omega\rightarrow[-1,+1]$ be the characteristic function of an algorithm A such that $\langle g,f_i\rangle\geq T$ for $i=1,2,...,d$. Then we have*

$$T\leq\sqrt{\frac{1+(d-1)\lambda}{d}}+\frac{70}{d}$$

for $d>100$.

Proof: We decompose g into basis as shown in equation 10 in lemma 9. Then we have

$$\begin{aligned}
T &\leq \langle g,f_i\rangle\\
&\leq \alpha_1\cdot\sqrt{1-\lambda}+\left(1-\sqrt{\frac{1-\lambda}{1+(d-1)\lambda}}\right)\sqrt{\frac{1+(d-1)\lambda}{d}}+\frac{60}{d}
\end{aligned}$$

for $i=1,2,...,d$.

Summing up these d inequalities, we get

$$\begin{aligned}
dT &\leq \sqrt{1-\lambda}\cdot\sum_{i=1}^{d}\alpha_i+d\cdot\left(1-\sqrt{\frac{1-\lambda}{1+(d-1)\lambda}}\right)\sqrt{\frac{1+(d-1)\lambda}{d}}+60\\
&\leq \sqrt{1-\lambda}\cdot\left(\sqrt{d}\cdot(1+\frac{50}{d})\right)+d\cdot\left(1-\sqrt{\frac{1-\lambda}{1+(d-1)\lambda}}\right)\sqrt{\frac{1+(d-1)\lambda}{d}}+60\\
&= \sqrt{1-\lambda}\cdot\sqrt{d}+d\cdot\left(1-\sqrt{\frac{1-\lambda}{1+(d-1)\lambda}}\right)\sqrt{\frac{1+(d-1)\lambda}{d}}+60+\sqrt{\frac{1-\lambda}{d}}\\
&\leq \sqrt{1+(d-1)\lambda}\cdot\sqrt{d}+70
\end{aligned}$$

or

$$T\leq\sqrt{\frac{1+(d-1)\lambda}{d}}+\frac{70}{d}$$

■

We next show that this bound is "almost tight", i.e., we give an example where $T=\sqrt{\frac{1+(d-1)\lambda}{d}}$.

Theorem 3 *For any odd prime p and any integer $n\geq 2$, there exists a class of $d=p^{n-1}$ boolean functions over GF_p^n: $\mathcal{F}=\{f_1,f_2,...,f_d\}$, and a distribution D, such that: any pair of the functions has identical inner product λ and the inner product of constant function $g(x)\equiv 1$ and any f_i is $\langle g,f_i\rangle_D=\sqrt{\frac{1+(d-1)\lambda}{d}}$.*

Proof: We consider the linear threshold functions:

$$f_{\vec{a},b}(\vec{x}) = \begin{cases} 1 & , \quad \text{if } \vec{a}\cdot\vec{x} \geq b \\ \\ -1 & , \quad \text{if } \vec{a}\cdot\vec{x} < b \end{cases}$$

where we fix b to be an element in GF_p between $(p-1)/4$ and $(p-1)/2]$, and let$\vec{a}$ be elements in GF_p^n with the restriction that $a^n = 1$: there are p^{n-1} of such functions.

We define the probabilistic distribution D to be

$$D(\vec{x}) = \begin{cases} 0 & \text{if } x^0 = x^1 = \cdots = x^{n-1} = 0 \\ \frac{1}{p^n - p} & \text{otherwise} \end{cases}$$

In other words, the distribution D is a uniform distribution over $GF_p^n - \{0^{n-1}0, 0^{n-1}1, ..., 0^{n-1}(p-1)\}$. The reason that we exclude the p elements $0^{n-1}0, 0^{n-1}1, ..., 0^{n-1}(p-1)$ is that all the functions behave the same on these p elements and thus it is trivial to predict their outputs. We use A to denote the set of these p elements:

$$A = \{0^{n-1}0, 0^{n-1}1, ..., 0^{n-1}(p-1)\}$$

Now we compute the inner product of $f_{\vec{a}_0,b}$ and $f_{\vec{a}_1,b}$. Notice that for any $\vec{a}_0, \vec{a}_1 \in GF_p^n$, $\vec{a}_0 \neq \vec{a}_1$, and $c_0, c_1 \in GF_p$, the equation

$$\begin{cases} \vec{a}_0 \cdot \vec{x} & = & c_0 \\ \vec{a}_1 \cdot \vec{x} & = & c_1 \end{cases}$$

always has p^{n-2} solutions for $\vec{x} \in GF_p^n$, provided that $a_0^n = a_1^n = 1$. We need to check the number of solutions that are in A: but if $\vec{x} \in A$, then

$$\vec{a}_0 \cdot \vec{x} = \vec{a}_1 \cdot \vec{x} = x^n$$

in which case there is exactly one solution in A, and thus there are $(p^{n-2} - 1)$ solutions in $GP_p^n - A$. Therefore

$$\begin{aligned}
\text{Pr}_D[f_{\vec{a}_0,b} = f_{\vec{a}_1,b} = -1] &= \sum_{c_0=0}^{b-1} \sum_{c_1=0}^{b-1} \text{Pr}_D[\vec{x}\cdot\vec{a}_0 = c_0 \ \wedge \ \vec{x}\cdot\vec{a}_1 = c_1] \\
&= \sum_{c=0}^{b-1} \text{Pr}_D[\vec{x}\cdot\vec{a}_0 = \vec{x}\cdot\vec{a}_1 = c] + \sum_{c_0=0}^{b-1} \sum_{c_1 \neq c_0} \text{Pr}_D[\vec{x}\cdot\vec{a}_0 = c_0 \ \wedge \ \vec{x}\cdot\vec{a}_1 = c_1] \\
&= \frac{1}{p^n - p}\left[b \cdot (p^{n-2} - 1) + b(b-1)\cdot p^{n-2}\right] \\
&= \frac{b^2 \cdot p^{n-2} - b}{p^n - p}
\end{aligned}$$

and similarly

$$\text{Pr}_D[f_{\vec{a}_0,b} = f_{\vec{a}_1,b} = 1] = \frac{(p-b)^2 \cdot p^{n-2} - (p-b)}{p^n - p}$$

So

$$\begin{aligned}
\text{Pr}_D[f_{\vec{a}_0,b} = f_{\vec{a}_1,b}] &= \text{Pr}_D[f_{\vec{a}_0,b} = f_{\vec{a}_1,b} = -1] + \text{Pr}_D[f_{\vec{a}_0,b} = f_{\vec{a}_1,b} = 1] \\
&= \frac{b^2 \cdot p^{n-2} - b}{p^n - p} + \frac{(p-b)^2 \cdot p^{n-2} - (p-b)}{p^n - p} \\
&= \frac{(p^2 - 2pb + 2b^2)\cdot p^{n-2} - p}{p^n - p}
\end{aligned}$$

Therefore we have

$$\begin{aligned}
\langle f_{\vec{a}_0,b}, f_{\vec{a}_1,b} \rangle &= \text{Pr}_D[f_{\vec{a}_0,b} = f_{\vec{a}_1,b}] - \text{Pr}_D[f_{\vec{a}_0,b} \neq f_{\vec{a}_1,b}] \\
&= 2 \cdot \text{Pr}_D[f_{\vec{a}_0,b} = f_{\vec{a}_1,b}] - 1 \\
&= 2 \cdot \frac{(p^2 - 2pb + 2b^2)\cdot p^{n-2} - p}{p^n - p} - 1 \\
&= \frac{(p-2b)^2 \cdot p^{n-2} - p}{p^n - p}
\end{aligned}$$

which we denote by λ.

Now let's compute the inner product of $f_{\vec{a},b}$ and g, which is the constant-one function.

$$\Pr{}_D[f_{\vec{a},b} = -1] \quad = \quad \sum_{c=0}^{b-1} \Pr{}_D[\vec{x} \cdot \vec{a} = c]$$

Notice that for any c, the equation $\vec{x} \cdot \vec{a} = c$ has exactly p^{n-1} solutions in GF_p^n, and exactly one of which is in A. Therefore

$$\Pr{}_D[f_{\vec{a},b} = -1] = \sum_{c=0}^{b-1} \Pr{}_D[\vec{x} \cdot \vec{a} = c] = b \cdot \frac{p^{n-1} - 1}{p^n - p} = \frac{b}{p}$$

and thus

$$\begin{aligned} \langle g, f_{\vec{a},b} \rangle &= \Pr{}_D[f_{\vec{a},b} = 1] - \Pr{}_D[f_{\vec{a},b} = -1] \\ &= 1 - 2 \cdot \Pr{}_D[f_{\vec{a},b} = -1] \\ &= \frac{p - 2b}{p} \end{aligned}$$

On the other hand, it is easy to verify that

$$\begin{aligned} \sqrt{\frac{1 + (d-1)\lambda}{d}} &= \sqrt{\frac{1 + (p^{n-1} - 1) \cdot \frac{(p-2b)^2 \cdot p^{n-2} - p}{p^n - p}}{p^{n-1}}} \\ &= \sqrt{\frac{(p-2b)^2}{p^2}} \\ &= \frac{p-2b}{p} = \langle g, f_{\vec{a},b} \rangle \end{aligned}$$

■

3.3 A Lower Bound for Statistical Query Learning

We have proved that without making any queries, a learning algorithm cannot learn a function family with advantage more than $\sqrt{\frac{1+(d-1)\lambda}{d}}$. Next we show that in order to improve the advantage, a lot of (normally exponentially many) queries have to be made. More precisely, we have the following theorem:

Theorem 4 *Let Ω be a domain of size M, let D be a probabilistic distribution over Ω. Let $\mathcal{F}$ be a class of functions $\mathcal{F} = \{f_1, f_2, ..., f_d\}$, such that $|\langle f_i, f_j \rangle - \lambda| \leq 1/d^3$ for all pairs $i \neq j$, where $1/2 \geq \lambda > 0$ and $d > 100$. Let A be an algorithm that makes Q queries to an honest SQ-oracle , each of which has sample count at most N, and learns $\mathcal{F}$ with advantage $S + \sqrt{\frac{1+(d-1)\lambda}{d}}$, where $S \geq d^{-1/4}$, then we have*

$$NQ \geq \frac{\sqrt{d} \cdot S}{2}$$

■

We comment that the total running time of A is bounded by NQ, since N is written in unary. Therefore thus the running time of A is also bounded by $\sqrt{d} \cdot S/2$. This theorem gives a tradeoff between the running time of A and the "extra" advantage it can have in learning $\mathcal{F}$: the running time goes up linearly with the advantage, and especially, to get a constant advantage, a running time of $\Omega(d^{1/2})$ is needed.

Proof: We assume A is a Turing Machine. Suppose the target function is f_j. We define the *state* of A after the k-th query to be the binary string S_k^j that describes the contents on A's tapes, the position of the heads, the current internal state of A. We define S_0^j to be the state of A before A starts. Notice each S_k^j is a random variable: the randomness comes from both the honest SQ-oracle and the random coins A tosses.

In the rest of the proof, we will omit the subscript A if there is no danger of confusion.

We define Δ_k to be the all-pair L_2 statistical distance among $S_k^1, S_k^2, ..., S_k^d$:

$$\Delta_k = \mathsf{SD}_2(S_k^1, S_k^2, ..., S_k^d)$$

Intuitively, Δ_k measures how "differently" A behaves when it has different target functions as inputs from the oracle.

We shall prove the following lemmas (in the appendix) considering the Δ_k's:

Lemma 10 $\Delta_0 = 0$.

This is obvious since A hasn't made any queries yet, and the state of A is independent of the target function.

Lemma 11 $\Delta_{k+1} - \Delta_k \leq N \cdot \sqrt{d}$

Proof: First we compute the all-pair SD among a special collection of random variables.

Let Q_j to be a random variable defined by

$$Q_j = g(X, f_j(X))$$

where g is a query function and X is distributed over Ω according to D. Notice Q_j can only be -1 or $+1$ for all j's, and thus we have

$$\begin{aligned}
\mathsf{SD}_2^2(Q_i, Q_j) &= (\mathsf{Pr}_D[Q_i = 1] - \mathsf{Pr}_D[Q_j = 1])^2 + (\mathsf{Pr}_D[Q_i = -1] - \mathsf{Pr}_D[Q_j = -1])^2 \\
&= 2 \cdot (\mathsf{Pr}_D[Q_i = 1] - \mathsf{Pr}_D[Q_j = 1])^2 \\
&= \frac{1}{2}(E_D[Q_i] - E_D[Q_j])^2
\end{aligned}$$

By lemma 6, we can write g in its Fourier coefficients:

$$g(x,y) = \sum_{i=1}^{M} \alpha_i \tilde{f}_i(x) + \sum_{i=1}^{M} \beta_i h(x,y)$$

and thus we have

$$g(x, f_j(x)) = \sum_{i=1}^{M} \alpha_i \tilde{f}_i(x) + \sum_{i=1}^{M} \beta_i \tilde{f}_i(x) f_j(y)$$

So the expected value of $g(x, f_j(x))$ is

$$E_D[g(x, f_j(x))] = E_D\left[\sum_{i=1}^{M} \alpha_i \tilde{f}_i(x) + \sum_{i=1}^{M} \beta_i \tilde{f}_i(x) \cdot f_j(x)\right]$$

and since we know from equation 4,

$$\tilde{f}_i(x) = \frac{1}{\sqrt{1-\lambda}} f_i(x) - \left(\frac{1}{\sqrt{1-\lambda}} - \frac{1}{\sqrt{1+(d-1)\lambda}}\right) \bar{f}(x)$$

we have

$$f_i(x) = \sqrt{1-\lambda}\tilde{f}_j(x) + \left(1 - \sqrt{\frac{1-\lambda}{1+(d-1)\lambda}}\right) \bar{f}(x)$$

and thus

$$\begin{aligned}
E_D[g(x, f_j(x))] &= \sum_{i=1}^{M} E_D[\alpha_i \tilde{f}_i(x)] + \sum_{i=1}^{M} E_D\left[\beta_i \tilde{f}_i(x) \cdot \left(\sqrt{1-\lambda}\tilde{f}_j(x) + \left(1 - \sqrt{\frac{1-\lambda}{1+(d-1)\lambda}}\right) \bar{f}(x)\right)\right] \\
&= C_g + \sqrt{1-\lambda} \cdot \sum_{i=1}^{d} \beta_i E_D[\tilde{f}_i(x) \tilde{f}_j(x)] \\
&= C_g + \sqrt{1-\lambda} \cdot \sum_{i=1}^{d} \beta_i \langle \tilde{f}_i, \tilde{f}_j \rangle
\end{aligned}$$

where

$$C_g = \sum_{i=1}^{M} E_D[\alpha_i \tilde{f}_i(x)] + \left(1 - \sqrt{\frac{1-\lambda}{1+(d-1)\lambda}}\right) \cdot \sum_{i=1}^{M} E_D[\beta_i \tilde{f}_i(x) \bar{f}(x)]$$

is a constant that is independent of the target function.

Therefore, if we look at the difference between $E_D[Q_1]$ and $E_D[Q_2]$, we have

$$
\begin{aligned}
|E_D[Q_1] - E_D[Q_2]| &= \sqrt{1-\lambda} \cdot \left| \sum_{i=1}^{d} \beta_i \left(\langle \tilde{f}_i, \tilde{f}_1 \rangle - \langle \tilde{f}_i, \tilde{f}_2 \rangle \right) \right| \\
&\leq \sum_{i=1}^{d} |\beta_i| \left(|\langle \tilde{f}_i, \tilde{f}_1 \rangle| + |\langle \tilde{f}_i, \tilde{f}_2 \rangle| \right) \\
&\leq |\beta_1 \left(\langle \tilde{f}_1, \tilde{f}_1 \rangle| + \langle \tilde{f}_2, \tilde{f}_1 \rangle \right) - \beta_2 \left(\langle \tilde{f}_1, \tilde{f}_2 \rangle + \langle \tilde{f}_2, \tilde{f}_2 \rangle \right) | + \sum_{i=3}^{d} |\beta_i| \left(\langle \tilde{f}_i, \tilde{f}_1 \rangle| + |\langle \tilde{f}_i, \tilde{f}_2 \rangle| \right) \\
&\leq |\beta_1 - \beta_2| + \frac{16}{d^3} \sum_{i=1}^{d} |\beta_i| \\
&\leq |\beta_1 - \beta_2| + \frac{24}{d^{5/2}}
\end{aligned}
$$

for $d > 100$. This result is true for any pair (Q_i, Q_j), and thus we have

$$
\begin{aligned}
\mathsf{SD}_2(Q_1, Q_2, ..., Q_d)^2 &= \sum_{i=1}^{d} \sum_{j=1}^{d} \mathsf{SD}_2(Q_i, Q_j)^2 \\
&= \frac{1}{2} \sum_{i=1}^{d} \sum_{j=1}^{d} (E_D[Q_i] - E_D[Q_j])^2 \\
&\leq \frac{1}{2} \sum_{i=1}^{d} \sum_{j=1}^{d} \left(|\beta_i - \beta_j| + \frac{24}{d^{5/2}} \right)^2 \\
&\leq \frac{1}{2} \sum_{i=1}^{d} \sum_{j=1}^{d} (\beta_i - \beta_j)^2 + \frac{1}{2} \sum_{i=1}^{d} \sum_{j=1}^{d} 2(|\beta_i| + |\beta_j|) \cdot \frac{24}{d^{5/2}} + \frac{1}{2} d^2 \cdot \left(\frac{24}{d^{5/2}} \right)^2 \\
&= \frac{1}{2} \sum_{i=1}^{d} \sum_{j=1}^{d} (\beta_i - \beta_j)^2 + 2d \cdot \sum_{i=1}^{d} |\beta_i| \cdot \frac{24}{d^{5/2}} + \frac{288}{d^3} \\
&\leq \frac{1}{2} \sum_{i=1}^{d} \sum_{j=1}^{d} (\beta_i - \beta_j)^2 + \frac{100}{d}
\end{aligned}
$$

Also,

$$
\begin{aligned}
\sum_{i=1}^{d} \sum_{j=1}^{d} (\beta_i - \beta_j)^2 &= d \cdot \sum_{i=1}^{d} \beta_i^2 - 2 \sum_{1 \leq i < j \leq d} \beta_i \beta_j \\
&\leq (d+1) \cdot \sum_{i=1}^{d} \beta_i^2 - \left(\sum_{i=1}^{d} \beta_i \right)^2 \\
&\leq (d+1) \cdot \sum_{i=1}^{d} \beta_i^2 \\
&\leq (d+1)(1 + \frac{100}{d})
\end{aligned}
$$

Putting things together, and we have

$$
\mathsf{SD}_2(Q_1, Q_2, ..., Q_d)^2 \leq \frac{1}{2}(d+1)(1 + \frac{100}{d}) + \frac{100}{d} \leq d
$$

Now, coming back to the Δ_k's: since we have sub-additivity for the all-pair L_2 statistical distances, and since the random bits used by the algorithm A are always the same regardless of the target function, the only place the statistical distance can increase is when A makes queries to the honest SQ-oracle. Furthermore, the amount of the increase is bounded by the SD among the replies returned by the oracle, which is $\mathsf{SD}_2(Q_1, Q_2, ..., Q_d)$. Notice that each query to the oracle has an sample count at most N, we have

$$
\Delta_{k+1} - \Delta_k \leq N \cdot \mathsf{SD}_2(Q_1, Q_2, ..., Q_d) \leq N \cdot \sqrt{d}
$$

■

Lemma 12 *$\Delta_Q \leq NQ \cdot \sqrt{d}$.*

Proof: This comes directly from lemma 10 and lemma 11. ■

Next we show the that in order to learn $\mathcal{F}$ with large advantage, the all-pair statistical distance has to be large.

Lemma 13 *If A learns $\mathcal{F}$ with advantage $S + \sqrt{\frac{1+(d-1)\lambda}{d}}$ where $S \geq d^{-1/4}$. Then $\Delta_Q \geq \frac{dS}{2}$.*

Proof: After making Q queries, the learning algorithm A should be ready to approximate the target function f_j. We write the characteristic function of A as ψ_j — notice this is a real-valued function since A might be a randomized algorithm and the honest SQ-oracle gives random answers. Also this function is dependent on the target function f_j.

On a particular input x, A outputs $+1$ with probability $(1+\psi_i(x))/2$ if the target function is f_i and $(1+\psi_j(x))/2$ if the target function is f_j. Therefore, when we look at the SD between S_Q^i and S_Q^j, we have

$$\mathsf{SD}_2(S_Q^i, S_Q^j)^2 \geq E_D\left[2 \cdot \left(\frac{1+\psi_i(x)}{2} - \frac{1+\psi_j(x)}{2}\right)^2\right] = \frac{1}{2}E_D[(\psi_i(x) - \psi_j(x))^2] \tag{12}$$

Now we do Fourier analysis for the ψ_j's: we write

$$\psi_j(x) = \sum_{k=1}^{M} \alpha_{jk}\tilde{f}_k(x)$$

then we have

$$\begin{aligned}
\mathsf{SD}_2(S_Q^i, S_Q^j)^2 &\geq \frac{1}{2}E_D[(\psi_i(x) - \psi_j(x))^2] \\
&= \frac{1}{2}E_D\left[\left(\sum_{k=1}^{M}(\alpha_{ik} - \alpha_{jk})\tilde{f}_k\right)^2\right] \\
&\geq \frac{1}{2}\sum_{k=1}^{d}(\alpha_{ik} - \alpha_{jk})^2 - \frac{16}{d}
\end{aligned}$$

Summing over all (i,j) pairs, we have

$$\begin{aligned}
\Delta_Q^2 &= \sum_{i=1}^{d}\sum_{j=1}^{d}\mathsf{SD}_2(S_Q^i, S_Q^j)^2 \\
&\geq \sum_{i=1}^{d}\sum_{j=1}^{d}\frac{1}{2}\sum_{k=1}^{d}(\alpha_{ik} - \alpha_{jk})^2 - \frac{16}{d} \\
&\geq \frac{1}{2}\sum_{k=1}^{d}\sum_{i=1}^{d}\sum_{j=1}^{d}(\alpha_{ik} - \alpha_{jk})^2 - 16d
\end{aligned}$$

On the other hand, since A learns $\mathcal{F}$ with advantage $S + \sqrt{\frac{1+(d-1)\lambda}{d}}$, we have

$$\langle \psi_j, f_j \rangle \geq S + \sqrt{\frac{1+(d-1)\lambda}{d}}, \quad j = 1, 2, ..., d$$

and by lemma 9,

$$S + \sqrt{\frac{1+(d-1)\lambda}{d}} \leq \langle g, f_j \rangle \leq \alpha_{jj} \cdot \sqrt{1-\lambda} + \left(1 - \sqrt{\frac{1-\lambda}{1+(d-1)\lambda}}\right)\sqrt{\frac{1+(d-1)\lambda}{d}} + \frac{60}{d}$$

or

$$\alpha_{jj} \geq \frac{1}{\sqrt{1-\lambda}}\left(S + \sqrt{\frac{1-\lambda}{d}} - \frac{60}{d}\right) \geq S$$

For each k, we define

$$A_k = \sum_{i=1}^{d}\alpha_{ik}^2$$

$$B_k = \max_{i=1}^{d} |\alpha_{ik}|$$

and

$$D_k = \sum_{i=1}^{d}\sum_{j=1}^{d}(\alpha_{ik} - \alpha_{jk})^2$$

So we have

$$B_k \geq \alpha_{kk} \geq S$$

Then

$$\begin{aligned}
C_k &= \sum_{i=1}^{d}\sum_{j=1}^{d}(\alpha_{ik} - \alpha_{jk})^2 \\
&\geq 2 \cdot \sum_{\alpha_{ik} \neq B_k} (B_k - \alpha_{ik})^2 \\
&= 2(d-1)B_k^2 - 4B_k \sum_{\alpha_{ik} \neq B_k} \alpha_{ik} + 2 \sum_{\alpha_{ik} \neq B_k} \alpha_{ik}^2 \\
&\geq 2(d-1)B_k^2 - 4B_k \left((d-1) \cdot \sum_{\alpha_{ik} \neq B_k} \alpha_{ik}^2 \right)^{\frac{1}{2}} + 2(A_k - B_k^2) \\
&= 2(d-1)B_k^2 - 4B_k \sqrt{(d-1)\cdot(A_k - B_k^2)} + 2(A_k - B_k^2) \\
&\geq 2(d-2)B_k^2 - \sqrt{d-1} \cdot A_k \\
&\geq 2(d-2)S^2 - \sqrt{d-1} \cdot A_k
\end{aligned}$$

where the first inequality is because of Cauchy-Schwartz and the second is because $B_k \cdot \sqrt{A_k - B_k^2} \leq (B_k^2 + A_k - B_k^2)/2 = A_k/2$.

Therefore

$$\begin{aligned}
\Delta_Q^2 &\geq \frac{1}{2}\sum_{k=1}^{d}\sum_{i=1}^{d}\sum_{j=1}^{d}(\alpha_{ik} - \alpha_{jk})^2 - 16d \\
&= \frac{1}{2}\sum_{k=1}^{d} C_k - 16d \\
&\geq d(d-2)S^2 - \frac{\sqrt{d-1}}{2}\sum_{k=1}^{d} A_k - 16d
\end{aligned}$$

Notice that when $d > 100$,

$$\sum_{k=1}^{d} A_k = \sum_{i=1}^{d}\sum_{k=1}^{d} \alpha_{ik}^2 \leq d \cdot (1 + \frac{100}{d}) = d + 100$$

by lemma 7. So

$$\Delta_Q^2 \geq d(d-2)S^2 - \frac{\sqrt{d-1}}{2}(d+100) - 16d \geq \frac{d^2 S^2}{4}$$

when $S \geq d^{-1/4}$.

■

Now putting lemma 12 and lemma 13 together, we have $NQ\sqrt{d} \geq \frac{dS}{2}$, or

$$NQ \geq \frac{\sqrt{d} \cdot S}{2}$$

■

Interesting, the techinique we use in this proof is similar to the one Ambainis uses in [A00] to prove a lower bound for the number of quantum queries a quantum search algorithm has to make.

As a comparison, implicit in [J00] is the following theorem:

Theorem 5 (Implicit in [J00]) *Let Ω be a domain of size M, let D be a probabilistic distribution over Ω. Let F be a class of functions $\mathcal{F} = \{f_1, f_2, ..., f_d\}$, such that $\langle f_i, f_j \rangle = 0$ for all pairs $i \neq j$. Let A be an algorithm that makes Q queries to an honest SQ-oracle, each of which has sample count at most N, and learns $\mathcal{F}$ with advantage S, then we have*

$$NQ = \Omega(d - \frac{1}{S^2})$$

Proof's sketch: Notice this is the case that all target functions are completely orthogonal and Fourier Analysis works perfectly. Suppose A has an advantage of S. Then the characteristic function ψ_A has an coefficient at least S for the target function. However ψ_A can have at most $1/S^2$ coefficients that are larger than or equal to S, by Parseval's equality. One can simply query $1/S^2$ more times to completely determine the target functions and have advantage 1. But as proved in [J00], $\Omega(d)$ queries are needed to learn $\mathcal{F}$ with advantage 1. Therefore we have $NQ = \Omega(d - 1/S^2)$. ■

The bound in [J00] is a bound for the specific case that all functions are orthogonal to each other, and in this case, it does give a better bound. Our bound is weaker, but it works for a more general class of functions.

3.4 Hardness for Learning Booleanized Linear Functions over a Finite Field

Next we show that the class of booleanized linear functions cannot be learned efficiently using statistical query:

Theorem 6 *Let p be an odd prime and $n > 1$ an integer. Let $\psi : GF_p \rightarrow \{-1, +1\}$ be a booleanizer such that $-\frac{1}{\sqrt{2}} \leq \langle \psi \rangle \leq \frac{1}{\sqrt{2}}$. and let $\mathcal{F}$ be a class of booleanized linear functions:*

$$\mathcal{F} = \{f_{\vec{a},\psi} \mid a^n = 1\}$$

Let D be the uniform distribution over GF_p^n. Then any algorithm with an access to an honest SQ-oracle that learns $\mathcal{F}$ with advantage $|\langle \psi \rangle| + S + 1/p^{n-1}$, where $S \geq p^{-(n-1)/4}$, with respect to distribution D, has a running time at least $p^{(n-1)/2} \cdot S/2$.

Proof: First notice there are totally p^{n-1} functions in $\mathcal{F}$. Second, for any $\vec{a}_1 \neq \vec{a}_2$, $a_1^n = a_2^n = 1$, and any $c_0, c_1 \in GF_p$, the equation

$$\begin{cases} \vec{a}_0 \cdot \vec{x} &= c_0 \\ \vec{a}_1 \cdot \vec{x} &= c_1 \end{cases}$$

will always have p^{n-2} solutions for $\vec{x} \in GF_p^n$.

Now we define set Ψ^- to be the set of negative examples for ψ and set Ψ^+ to be the set of positive examples for ψ. In other words,

$$\Psi^- = \{x \in GF_p \mid \psi(x) = -1\}, \quad \Psi^+ = \{x \in GF_p \mid \psi(x) = +1\}$$

We also define

$$s_0 = |\Psi^-|, \quad s_1 = |\Psi^+|$$

then we have $s_0 + s_1 = p$ and $\langle \psi \rangle = (s_1 - s_0)/p$. Therefore,

$$\begin{aligned} \Pr[f_{\vec{a}_0,b} = f_{\vec{a}_1,b} = -1] &= \sum_{c_0 \in \Psi^-} \sum_{c_1 \in \Psi^-} \Pr[\vec{x} \cdot \vec{a}_0 = c_0 \ \wedge \ \vec{x} \cdot \vec{a}_1 = c_1] \\ &= s_0^2 \cdot \frac{p^{n-2}}{p^n} \\ &= \frac{s_0^2}{p^2} \end{aligned}$$

and similarly,

$$\begin{aligned} \Pr[f_{\vec{a}_0,b} = f_{\vec{a}_1,b} = +1] &= \sum_{c_0 \in \Psi^+} \sum_{c_1 \in \Psi^+} \Pr[\vec{x} \cdot \vec{a}_0 = c_0 \ \wedge \ \vec{x} \cdot \vec{a}_1 = c_1] \\ &= \frac{s_1^2}{p^2} \end{aligned}$$

So

$$\Pr[f_{\vec{a}_0,b} = f_{\vec{a}_1,b}] = \Pr[f_{\vec{a}_0,b} = f_{\vec{a}_1,b} = -1] + \Pr[f_{\vec{a}_0,b} = f_{\vec{a}_1,b} = +1] = \frac{s_0^2 + s_1^2}{p^2}$$

and thus

$$\langle f_{\vec{a}_0,b}, f_{\vec{a}_1,b}\rangle = 2 \cdot \Pr\left[f_{\vec{a}_0,b} = f_{\vec{a}_1,b}\right] - 1 = \left(\frac{s_1 - s_0}{p}\right)^2$$

If we let $d = p^{n-1}$ and $\lambda = \left(\frac{s_1-s_0}{p}\right)^2$, then by theorem 4, we know that any algorithm that learns $\mathcal{F}$ by statistical query with an advantage $S + \sqrt{\frac{1+(d-1)\lambda}{d}}$ will have a running time at least $\sqrt{d} \cdot S/2 = p^{(n-1)/2} \cdot S/2$. Notice that

$$\begin{aligned}
\sqrt{\frac{1+(d-1)\lambda}{d}} &= \left(\lambda + \frac{1-\lambda}{d}\right)^{\frac{1}{2}} \\
&\leq \sqrt{\lambda} \cdot \left(1 + \frac{1}{2} \cdot \frac{1-\lambda}{d \cdot \sqrt{\lambda}}\right) \\
&\leq \sqrt{\lambda}(1 + \frac{1}{d}) \\
&\leq \left|\frac{s_1 - s_0}{p}\right| + \frac{1}{p^{n-1}} \\
&= |\langle\psi\rangle| + \frac{1}{p^{n-1}}
\end{aligned}$$

So an algorithm with advantage $|\langle\psi\rangle| + S + 1/p^{n-1}$ should have running time $p^{(n-1)/2} \cdot S/2$.

■

Notice that it is not hard to prove that each function $f_{\vec{a},\psi}$ has bias $\langle\psi\rangle$. If $\langle\psi\rangle \geq 0$, then the constant function $g(x) = +1$ already has an advantage $\langle\psi\rangle$ in approximating $\mathcal{F}$, otherwise $g(x) = -1$ has an advantage $\langle\psi\rangle$ in approximating $\mathcal{F}$.

This result gives some positive evidence towards the security of the private-key cryptosystem proposed by Baird [B01].

Since linear threshold functions are special cases for booleanized linear functions, we have the following theorem:

Theorem 7 *Let p be an odd prime and $n > 1$ an integer. Let b be a non-zero element in GF_p such that $(p-1)/4 \leq b \leq 3(p-1)/4$, and let $\mathcal{F}$ be a class of linear threshold functions:*

$$\mathcal{F} = \{f_{\vec{a},b} \mid a^n = 1\}$$

Let D be the uniform distribution over GF_p^n. Then any algorithm with an access to an honest SQ-oracle that learns $\mathcal{F}$ with advantage $|(p-2b)/p| + S + 1/p^{n-1}$, where $S \geq p^{-(n-1)/4}$, with respect to distribution D, has a running time at least $p^{(n-1)/2} \cdot S/2$.

Furthermore, we have:

Corollary 1 *For the class of linear threshold functions, in the case that p is exponentially large in n and $b = (p+1)/2$, no statistical query algorithm can weakly learn $\mathcal{F}$.*

Proof: When $b = (p+1)/2$, we have $(p-2b)/p = -1/p$, which is exponentially small in n. If an algorithm A weakly learns $\mathcal{F}$, it has to have an advantage $\epsilon > \frac{1}{n^c}$ for some constant c. Then by theorem 7 the running time of A has to be at least $p^{(n-1)/2} \cdot (\epsilon - 1/p - 1/p^{n-1})$, which is exponentially large in n. ■

4 Algorithm for Linear Threshold Functions

In this section we present an algorithm BUILD-TREE that learns the special class of linear threshold functions as shown in corollary 1, using a random example oracle. The running time of BUILD-TREE is slightly better than the brute-force algorithm, and also slightly better than the lower bound for the statistical query model.

We first state the problem: pick an integer $n > 1$ and an odd prime p such that p is exponentially large in n. Let $b = (p+1)/2$, and the class of functions is the class of linear threshold functions with the fixed b and with the constraint that the n-th entry of $\vec{a}$ is 1:

$$\mathcal{F} = \{f_{\vec{a}} \mid a^n = 1\}^2$$

The distribution over the inputs is the uniform distribution. We show an algorithm that learns any function $f \in \mathcal{F}$ in time $p^{O(n/\log n)}$, with advantage 0.5, with respect to a random example oracle. Notice the brute-force algorithm that examines all possible functions has running time $p^{O(n)}$ and any SQ-algorithm much also have a running time $p^{O(n)}$ to have a constant advantage in learning $\mathcal{F}$.

[2]In the rest of this section, we omit b in the description of $f_{\vec{a},b}$ since b is fixed to be $(p+1)/2$.

4.1 Description of the Algorithm

The idea for BUILD-TREE is pretty intuitive: given a target function $f_{\vec{a}}$, we know there is a "secret vector" $\vec{a}$ associated with the function. If one picks a *random* negative example $\vec{x}$, then the *expected* value of $\vec{a} \cdot \vec{x}$ is $(p-1)/4$. If we draw $(4q+1)$ random negative samples, the expected sum of the inner products is about $(4q+1)(p-1)/4$, which is about $(p-1)/4$ modulo p, if $q \ll p$ (in our algorithm, we have $q = O(\log n) = O(\log\log p)$).. So it is more likely that the sum of $(4q+1)$ random negative examples is still a negative example than is a positive one. The algorithm exploits this "marginal difference", boosts it by Chernoff bound, and gains a constant advantage in learning $\mathcal{F}$. What the algorithm does is: it draws a lot $(p^{O(n/\log n)}$ negative examples, and when getting an input $\vec{X}$, BUILD-TREE tries to write $\vec{X}$ as the sum of $(4q+1)$ negative examples it drew, and estimates the success probability. If the success probability is high, it outputs "$f(\vec{X}) = -1$", otherwise it outputs "$f(\vec{X}) = +1$". The name of the algorithm comes from the fact that the algorithm estimates the probability by building a complete binary tree from the samples it draws.

Our algorithm is inspired by the algorithm Blum et al. used in [BKW00] to learn noisy parity functions, where the main idea is also trying to write an input as logarithmically many samples.

Now we describe BUILD-TREE in more detail:

The algorithm BUILD-TREE has a random example oracle EX_f, which, at each invocation, produces a random pair $(\vec{x}, f_{\vec{a}}(\vec{x}))$, where $\vec{x}$ is uniformly chosen from GF_p^n. The algorithm also has an input $\vec{X}$ on which it tried to predict $f_{\vec{a}}(\vec{X})$.

The algorithm consists of 2 phases. In Phase I, it draws about $p^{O(n/\log n)}$ samples and processes them; in Phase II, it reads the input $\vec{X}$ and tries to build a complete binary tree from the samples it drew in Phase I, where each node is a multi-set of elements in GF_p^n. Finally, BUILD-TREE counts the number of elements in the root node and use this number to predict $f_{\vec{a}}(\vec{X})$.

- **Phase I:** We define $a = \log n/2$ and $b = 2n/\log n$, and think of each vector in GF_p^n as divided into a blocks, each block containing b elements in GF_p.

  We define
  $$K = p^b \cdot 2^{2^{a+1}} \cdot n$$
  BUILD-TREE draws $2^{a-1}(2^a+1)K$ negative samples. Notice each $f_{\vec{a}}$ is "reasonably balanced" and thus there would be no trouble getting a lot of negative samples. We use N to denote $2^{a-1}K$. BUILD-TREE groups these samples into 2^a+1 groups of N elements each, and denotes these groups by $G_0, G_1, ..., G_{2^a}$. Then it add the last 2 groups G_{2^a-1} and G_{2^a} entry-wise to form a new group, G'_{2^a-1}. More precisely, suppose $G_{2^a-1} = \{a_1, a_2, ..., a_N\}$ and $G_{2^a-1} = \{b_1, b_2, ..., b_N\}$, then
  $$G'_{2^a-1} = \{a_1+b_1, a_2+b_2, ..., a_N+b_N\}$$
  is also a group of N numbers. Now define $G'_i = G_i$, for $i = 0, 1, ..., 2^a-2$, and now we have 2^a groups $G'_1, ..., G'_{2^a-1}$ of N elements.

- **Phase II:** In this phase BUILD-TREE gets a new sample $\vec{X}$ and it tries to learn $f_{\vec{a}}(\vec{X})$. The approach is to try to write $\vec{X}$ as the sum of 2^a+1 negative samples drawn from phase I. More precisely BUILD-TREE tries to find 2^a elements $\vec{x}_1, \vec{x}_2, ..., \vec{x}_{2^a-1}$, such that $\vec{x}_i \in G'_i$ for $i = 1, 2, ..., 2^a-1$ and
  $$\vec{X} = \vec{x}_1 + \vec{x}_2 + \cdots + \vec{x}_{2^a-1}.$$
  Notice $\vec{x}_{2^a-1} \in G'_{2^a-1}$ is already a sum of 2 negative samples, and thus if one can find such an $\vec{X}$, it is the sum of 2^a+1 negative samples.

  Since BUILD-TREE is working in GF_p, it can compute $\vec{Y} = \frac{1}{2^a}\vec{X}$, and subtract $\vec{Y}$ from each element in each group G'_i. More precisely, we define
  $$A_i = \{\vec{x} - \vec{Y} \mid \vec{x} \in G'_i\}$$
  for $i = 0, 2, ..., 2^a-1$. Then the task for BUILD-TREE becomes finding 2^a elements, one from each A_i such that they add up to $\vec{0}$.

  To do so, BUILD-TREE will build a complete binary tree of sets. First some notations: We define the *height* of a node in a binary tree as the shortest distance from this node to a leaf node, and a leaf node has height 0. The height of a binary tree is the height of its root. A node that is neither a leaf node nor the root node is called an *internal node*. There are (2^a-2) internal nodes for a complete binary tree of height a.

  The BUILD-TREE algorithm will build a complete binary tree of height a, and every node in the tree is a multi-set of elements in GF_p^n: on the leaves are the sets A_i, and each internal node is a set whose elements are sums of the elements of its two children nodes. All internal nodes of height l contain $2^{a-l-1}K$ elements, all of which have 0's at the first l blocks.

Here is the actual construction:

BUILD-TREE will build a complete binary tree of height a, and there are 2^{a-k} nodes of height k: we will denote these nodes by $G_0^k, G_1^k, ..., G_{2^{a-k}-1}^k$. The construction is from bottom-up: one builds the nodes of height 0, or the leaf nodes first, and then the node of height $1, 2, ..., a-1$, and finally the root node.

- **LEAF NODES:**
  The leaf nodes are just the sets $A_0, A_1, ..., A_{2^a-1}$. In other words, let $G_i^0 = A_i$ for $i = 0, 1, ..., 2^a - 1$.
- **INTERNAL NODES:**
  After all the nodes of height $(l-1)$ are built, BUILD-TREE constructs the nodes of height l.
  To construct node G_i^l, BUILD-TREE needs nodes G_{2i}^{l-1} and G_{2i+1}^{l-1}, namely, the two children nodes of G_i^l. The BUILD-TREE does the following:
  It starts by setting G_i^l to be the empty set and label all elements in G_{2i}^{l-1} and G_{2i+1}^{l-1} as "unmarked".
  It repeats the following "SELECT-AND-MARK" process for $2^{a-l-1}K$ times:
  BEGIN OF SELECT-AND-MARK
  * BUILD-TREE (arbitrarily) picks an unmarked element $\vec{u} \in G_{2i}^{l-1}$, and scans G_{2i+1}^{l-1} to check if there is an unmarked element $\vec{v} \in G_{2i+1}^{l-1}$, such that $\vec{u} + \vec{v}$ has the first l blocks all-zero. Notice that both $\vec{u}$ and $\vec{v}$ has the first $l-1$ blocks all-zero already, and thus BUILD-TREE is actually looking for a $\vec{v}$ whose l-th block is the complement of that of $\vec{u}$.
  * If BUILD-TREE finds such a $\vec{v}$, it puts $\vec{u} + \vec{v}$ into G_i^l and marks both $\vec{u}$ and $\vec{v}$.
  * If BUILD-TREE can't find such a $\vec{v}$, it aborts: the algorithm fails.

  END OF SELECT-AND-MARK
  If BUILD-TREE doesn't abort in the $2^{a-l-1}K$ SELECT-AND-MARK processes, it constructs a set G_i^l of size $2^{a-l-1}K$.
- **ROOT NODE:**
  If BUILD-TREE doesn't abort in constructing the $(2^a - 2)$ internal nodes, it proceeds to build the root node, G_0^a. Notice the children of node G_0^a are nodes G_0^{a-1} and G_1^{a-1}, each of which contains K elements: suppose that
  $$G_0^{a-1} = \{\vec{u}_1, \vec{u}_2, ..., \vec{u}_K\}$$
  and
  $$G_1^{a-1} = \{\vec{v}_1, \vec{v}_2, ..., \vec{v}_K\}$$
  Then the root node G_0^a is
  $$G_0^a = \{\vec{u}_i + \vec{v}_i \mid \vec{u}_i + \vec{v}_i = \vec{0},\ i = 1, 2, ..., K\}$$
  In other words, G_0^a is a multi-set of $\vec{0}$'s, and the size of G_0^a depends on the number of corresponding pairs of vectors in G_0^{a-1} and G_1^{a-1} that are complement to each other.

In this way BUILD-TREE builds a complete binary tree all the way up the the root. If the size of the root node is greater than $2^{2^{a+1}} \cdot n$, BUILD-TREE outputs "$f(X) = -1$"; otherwise is outputs "$f(X) = +1$".

4.2 Analysis of the BUILD-TREE Algorithm

We fix the target function $f_{\vec{a}}$, and define two sets $A = \{0, 1, ..., (p-1)/2\}$ and $B = \{(p+1)/2, (p+1)/2, ..., p-1\}$. A and B forms a partition of GF_p and an $\vec{x}$ is a negative example, if and only if $\vec{a} \cdot \vec{x} \in A$.

Lemma 14 *For any $l = 0, 1, ..., (a-1)$, the elements in the nodes of height l are independent to each other.*

Proof: Notice each element in nodes of height l is a sum of 2 elements in nodes of height $l-1$, and is recursively a sum of 2^l elements in the leaf nodes. Since BUILD-TREE marks all the elements that are "used", different elements in nodes of height l are sums of totally different elements of the leaf nodes. The elements in the leaf nodes are independent to each other, and thus the elements in the nodes of height l are also independent to each other. ■

Lemma 15 *For each randomly chosen negative example $\vec{x} = (x^1, x^2, ..., x^n)$, if we only look at its prefix, $(x^1, x^2, ..., x^{n-1})$, they are uniformly randomly distributed.*

Proof: Notice the set of negative samples is the set of $\vec{x}$'s such that $\vec{a} \cdot \vec{x} \in A$. For any particular setting of $(x^1, x^2, ..., x^{n-1})$, there are exactly $|A| = (p-1)/2$ elements $y \in GF_p$ such that $\vec{a} \cdot (x^1, x^2, ..., x^{n-1}, y) \in A$, since we have $a^n = 1$.

Therefore each prefix $(x^1, x^2, ..., x^{n-1})$ appears exactly the same number of times, and thus these prefixes are uniformly distributed. ■

Lemma 16 *For any element in nodes of height l, $0 \le l < a$, the $(l+1), ..., (a-1)$-th blocks of the element are uniformly distributed.*

Proof: We prove by induction. The case $l = 0$ is obvious from lemma 15. For $l > 0$, notice an element in a node of height l is constructed by adding two elements whose l-th block are complement to each other — we denote these two elements of height $(l-1)$ by $\vec{u}$ and $\vec{v}$. By inductive hypothesis, the $(l+1), (l+2), ..., (a-1)$-th blocks of both $\vec{u}$ and $\vec{v}$ are independently uniformly distributed, and since the construction of a node of height l doesn't use any information about the $(l+1), (l+2), ..., (a-1)$-th blocks, the $(l+1), (l+2), ..., (a-1)$-th blocks of $\vec{u} + \vec{v}$ are still uniformly distributed. ∎

Lemma 17 *In constructing the binary tree, for any internal node, the probability that the construction aborts is at most $2^a \cdot e^{-n}$.*

Proof: Notice that if we look at the first $a-1$ blocks, all the samples are like elements uniformly chosen from GF_p^n. Furthermore, the information about the last block is only needed in constructing the root node. So when we compute the probability that BUILD-TREE aborts, it doesn't make any difference if we "pretend" all our samples are chosen uniformly from GF_p^n, instead of chosen uniformly from the the set of negative examples. Notice in constructing the node G_i^l, BUILD-TREE aborts only if for a particular $\vec{u} \in G_{2i}^{l-1}$, none of the unmarked elements in G_{2i+1}^{l-1} has the l-th block that is the complement of the l-th block of $\vec{u}$. But there are always at least $K = p^b \cdot 2^{2^{a+1}} \cdot n$ unmarked elements in G_{2i+1}^{l-1}, each of which is independently uniformly distributed, and there are only p^b possible values for the l-th block, then probability that BUILD-TREE aborts for this node is at most

$$(1 - \frac{1}{p^b})^{p^b \cdot 2^{2^{a+1}} \cdot n} < e^{-n}$$

And since BUILD-TREE constructs totally $2^a - 2$ internal nodes, the probability that it aborts is at most $2^a \cdot e^{-n}$ ∎

We next compute the expected size of the root node, which is K times the probability that an element from G_0^{a-1} and an element from G_1^{a-1} add up to $\vec{0}$.

Lemma 18 *Suppose $\vec{a} \cdot \vec{X} = l$. The probability that an element from G_0^{a-1} and an element from G_1^{a-1} add up to $\vec{0}$ equals the $1/p^{b-1}$ times the probability that the sum of $2^a + 1$ randomly chosen elements from A equals l.*

Proof: We use P_l to denote the probability that $2^a + 1$ random elements from A add up to l, or

$$P_l = \Pr{}_{r_1, r_2, \ldots, r_{2^a+1} \in A}[r_1 + r_2 + \cdots + r_{2^a+1} = l]$$

and we need to prove that

$$\Pr[\vec{x} + \vec{y} = \vec{0} \mid \vec{x} \in G_0^{a-1}, \vec{y} \in G_1^{a-1}] = \frac{P_l}{p^{b-1}}$$

For an arbitrary pair of elements $\vec{x} \in G_0^{a-1}$ and $\vec{y} \in G_1^{a-1}$, we write them as

$$\vec{x} = \vec{x}_0 + \vec{x}_1 + \cdots + \vec{x}_{2^{a-1}-1} - \frac{1}{2}\vec{X}$$

and

$$\vec{y} = \vec{x}_{2^{a-1}} + \vec{x}_{2^{a-1}+1} + \cdots + \vec{x}_{2^a} - \frac{1}{2}\vec{X}$$

Where each $\vec{x}_i$ and is a random negative sample.

We define $r_i = \vec{x}_i \cdot \vec{a}$ for $i = 0, 1, ..., 2^a$. Then we have $r_i \in A$ for all i's. It is useful to view $\vec{x}_i$ as being chosen in the following two-step procedure: first one chooses a random $r_i \in A$ and then randomly chooses an $\vec{x}_i \in GF_p^n$ such that $\vec{x}_i \cdot \vec{a} = r_i$ — notice when r_i is fixed, choosing an $\vec{x}_i \in GF_p^n$ such that $\vec{x}_i \cdot \vec{a} = r_i$ is equivalent to randomly choosing the first $n-1$ entries of $\vec{x}_i$ and setting x_i^n accordingly.

Now we fix all the r_i's and consider the probability that $\vec{x} + \vec{y} = \vec{0}$, *conditioned on that $\vec{x}_i \cdot \vec{a} = r_i$, $i = 0, 1, ...2^a$.*

Notice that we have

$$(\vec{x} + \vec{y}) \cdot \vec{a} = \sum_{i=0}^{2^a} \vec{x}_i \cdot a - \vec{X} \cdot \vec{a} = \sum_{i=0}^{2^a} r_i - l \tag{13}$$

and thus if $l \neq \sum_{i=0}^{2^a} r_i$, the probability that $\vec{x} + \vec{y} = \vec{0}$ is 0.

Now we consider the case that $l = \sum_{i=0}^{2^a} r_i$ — we shall prove that in this case, the probability $\vec{x} + \vec{y} = \vec{0}$ is always $1/p^{b-1}$.

Notice that $\vec{x} + \vec{y} = \vec{0}$ is actually a set of b linear equations: the last b entries of $\vec{x} + \vec{y}$ are all zero. But conditioned on $\vec{x}_i \cdot \vec{a} = r_i$ and $l = \sum_{i=0}^{2^a} r_i$, these b equations are not independent: the first $(b-1)$ equations will imply the last one, since we have equation 13 and since $a^n = 1$. In other words, in computing the probability that $\vec{x}+\vec{y}=\vec{0}$, we don't have to consider the last entry of $\vec{x}_i$'s since they are already "taken care of" by the conditions $\vec{x}_i \cdot \vec{a} = r_i$ and $l = \sum_{i=0}^{2^a} r_i$. However, by lemma 16, if we don't consider the last entries, all the $\vec{x}_i$'s are uniformly distributed, and so is $\vec{x}+\vec{y}$, which is the sum of 2^a+1 uniformly distributed vectors minus $\vec{X}$, and is still a uniform vector in GF_p^{n-1}. So the probability that $\vec{x}+\vec{y}=\vec{0}$ is $\frac{1}{p^{n-1}}$ since there are p^{n-1} possible values for $\vec{x}+\vec{y}$.

Finally we have

$$\begin{aligned}\mathsf{Pr}\,[\vec{x}+\vec{y}=\vec{0} \mid \vec{x}\in G_0^{a-1}, \vec{y}\in G_1^{a-1}] &= \mathsf{Pr}\,[\vec{x}+\vec{y}=\vec{0} \mid \sum_{i=0}^{2^a} r_i = l]\cdot \mathsf{Pr}\,[\sum_{i=0}^{2^a} r_i = l] \\ &= \frac{P_l}{p^{n-1}}\end{aligned}$$

■

So the probability that the construction of the binary tree succeeds only depends on l, the inner product of the input $\vec{X}$ and the secret vector $\vec{a}$. Next we give the bounds for P_l, for different l.

Lemma 19 *Let m be a natural number such that $m \equiv 1 \pmod 4$ and $m > 10$. Let p be an odd prime number ($p \geq 2$) that is larger than $50m$. Let l be a natural number such that $0.05p < l < 0.45p$. Let A be the set $\{0, 1, ..., (p-1)/2\}$. Let $x_1, x_2, ..., x_m$ be random samples chosen uniformly from A and let $y = x_1 + x_2 + ... x_m \pmod p$, Then*

$$\mathsf{Pr}\,[y = l] > \frac{1}{p}(1 + 0.5^m)$$

Proof: We use k to denote $(p-1)/2$, and thus $A = \{0, 1, ..., k\}$. We look at a polynomial

$$F(x) = (1 + x + \cdots + x^k)^m$$

We can expand out this polynomial modulo $(x^p - 1)$ and get

$$F(x) \equiv \alpha_0 + \alpha_1 \cdot x + \cdots \alpha_{p-1} \cdot x^{p-1} \pmod{(x^p - 1)}$$

It is easy to see that

$$\mathsf{Pr}\,[y \in l] = \frac{\alpha_l}{(k+1)^m} \tag{14}$$

Now we focus on computing these α_j's.

We define $\omega = e^{i\frac{2\pi}{p}}$ to be a p-th root of 1. If we plug ω^j into $F(x)$, we get

$$\sum_{l=0}^{p-1} \alpha_l \cdot \omega^{jl} = F(\omega^j)$$

So we have a linear system: define a $p \times p$ matrix U as:

$$U = \begin{pmatrix} 1 & \omega & \cdots & \omega^{p-1} \\ \omega & \omega^2 & \cdots & \omega^{2(p-1)} \\ \vdots & \vdots & \ddots & \vdots \\ \omega^{p-1} & \omega^{2(p-1)} & \cdots & \omega^{(p-1)^2} \end{pmatrix}$$

In general, $[U]_{jl} = \omega^{jl}$ — in fact, U is the Fourier Transformation matrix, and we have

$$U \cdot \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_{p-1} \end{pmatrix} = \begin{pmatrix} F(1) \\ F(\omega) \\ \vdots \\ F(\omega^{p-1}) \end{pmatrix}$$

We define $V = U^{-1}$ and it is easy to verify that $[V]_{jl} = \frac{1}{p} \cdot U^{-jl}$, or

$$V = U^{-1} = \frac{1}{p} \begin{pmatrix} 1 & \omega^{-1} & \cdots & \omega^{-(p-1)} \\ \omega & \omega^{-2} & \cdots & \omega^{-2(p-1)} \\ \vdots & \vdots & \ddots & \vdots \\ \omega^{-(p-1)} & \omega^{-2(p-1)} & \cdots & \omega^{-(p-1)^2} \end{pmatrix}$$

So we have

$$\begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_{p-1} \end{pmatrix} = V \cdot \begin{pmatrix} F(1) \\ F(\omega) \\ \vdots \\ F(\omega^{p-1}) \end{pmatrix} = \frac{1}{p} \begin{pmatrix} 1 & \omega^{-1} & \cdots & \omega^{-(p-1)} \\ \omega & \omega^{-2} & \cdots & \omega^{-2(p-1)} \\ \vdots & \vdots & \ddots & \vdots \\ \omega^{-(p-1)} & \omega^{-2(p-1)} & \cdots & \omega^{-(p-1)^2} \end{pmatrix} \cdot \begin{pmatrix} F(1) \\ F(\omega) \\ \vdots \\ F(\omega^{p-1}) \end{pmatrix}$$

or

$$\alpha_l = \frac{1}{p} \sum_{j=0}^{p-1} \omega^{-jl} F(j) \tag{15}$$

So we have

$$\Pr[y = l] = \frac{\alpha_l}{(k+1)^m} = \frac{1}{p(k+1)^m} \cdot \sum_{j=0}^{p-1} \omega^{-jl} F(j) \tag{16}$$

Notice that

$$F(\omega^j) = (1 + \omega^j + \cdots + \omega^{jk})^m = \begin{cases} (k+1)^m & , \quad j = 0 \\ \left(\frac{1-\omega^{j(k+1)}}{1-\omega^j}\right)^m & , \quad \text{otherwise} \end{cases} \tag{17}$$

Substituting in the formula for $F(j)$ in equation 17, we have

$$\Pr[y = l] = \frac{1}{p(k+1)^m} \cdot \sum_{j=0}^{p-1} \omega^{-jl} F(j) = \frac{1}{p} + \frac{1}{p(k+1)^m} \cdot \sum_{j=1}^{p-1} \omega^{-jl} \cdot \left(\frac{1-\omega^{j(k+1)}}{1-\omega^j}\right)^m \tag{18}$$

Again, notice that $\omega^j = \omega^{2j(k+1)}$, we have

$$\begin{aligned} \Pr[y = l] &= \frac{1}{p} + \frac{1}{p(k+1)^m} \cdot \sum_{j=1}^{p-1} \frac{\omega^{-jl}}{(1+\omega^{j(k+1)})^m} \\ &= \frac{1}{p} + \frac{1}{p(k+1)^m} \cdot \sum_{j=1}^{p-1} \frac{\omega^{-2l(j+1)k}}{(1+\omega^{j(k+1)})^m} \\ &= \frac{1}{p} + \frac{1}{p(k+1)^m} \cdot \sum_{j=1}^{p-1} \frac{\omega^{-2lj}}{(1+\omega^j)^m} \end{aligned}$$

We notice that

$$\frac{\omega^{-2l(-j)}}{(1+\omega^{-j})^m} = \overline{\left(\frac{\omega^{-2lj}}{(1+\omega^j)^m}\right)}$$

and if we sum over these two terms, we get

$$\frac{\omega^{-2l(-j)}}{(1+\omega^{-j})^m} + \frac{\omega^{-2lj}}{(1+\omega^j)^m} = \frac{\omega^{2lj}(1+\omega^j)^m + \omega^{-2lj}(1+\omega^{-j})^m}{|1+\omega^j|^{2m}}$$

Notice that $\omega = e^{i\frac{2\pi}{p}}$ and thus

$$1 + \omega^j = 1 + e^{i\frac{2\pi j}{p}} = 2\cos(\frac{j\pi}{p}) \cdot e^{i\frac{j\pi}{p}}$$

Therefore

$$\begin{aligned} \omega^{2lj}(1+\omega^j)^m &= e^{i\frac{4lj\pi}{p}} \cdot \left(2\cos(\frac{j\pi}{p}) \cdot e^{i\frac{j\pi}{p}}\right)^m \\ &= \left(2\cos(\frac{j\pi}{p})\right)^m \cdot e^{i\frac{(m+4l)j}{p}} \end{aligned}$$

and similarly

$$\begin{aligned} \omega^{-2lj}(1+\omega^{-j})^m &= e^{-i\frac{4lj\pi}{p}} \cdot \left(2\cos(\frac{j\pi}{p}) \cdot e^{-i\frac{j\pi}{p}}\right)^m \\ &= \left(2\cos(\frac{j\pi}{p})\right)^m \cdot e^{-i\frac{(m+4l)j}{p}} \end{aligned}$$

Summing them up, and we get

$$\frac{\omega^{-2l(-j)}}{(1+\omega^{-j})^m} + \frac{\omega^{-2lj}}{(1+\omega^{j})^m} \quad = \quad 2 \cdot \frac{\cos(\frac{(m+4l)j\pi}{p})}{\left(2\cos(\frac{j\pi}{p})\right)^m}$$

Now we can simplify the formula for the probability in equation 18 to:

$$\mathsf{Pr}\,[y=l] = \frac{1}{p} + \frac{1}{p(k+1)^m} \cdot \sum_{j=1}^{p-1} \frac{\cos(\frac{(m+4l)j\pi}{p})}{\left(2\cos(\frac{j\pi}{p})\right)^m} \tag{19}$$

Now since we require m to be 1 modulo 4, we write m as $m = 4q+1$. Also since m is an odd number, it is easy to verify that

$$\frac{\cos(\frac{(m+4l)j\pi}{p})}{\left(2\cos(\frac{j\pi}{p})\right)^m} = \frac{\cos(\frac{(m+4l)(p-j)\pi}{p})}{\left(2\cos(\frac{(p-j)\pi}{p})\right)^m}$$

If we define $r = (p-1)/2 - j$, we know that

$$\begin{aligned}
\mathsf{Pr}\,[y=l] &= \frac{1}{p} + \frac{1}{p(k+1)^m} \cdot \sum_{j=1}^{p-1} \frac{\cos(\frac{(m+4l)j\pi}{p})}{\left(2\cos(\frac{j\pi}{p})\right)^m} \\
&= \frac{1}{p} + \frac{2}{p(k+1)^m} \cdot \sum_{j=1}^{(p-1)/2} \frac{\cos(\frac{(4q+1+4l)j\pi}{p})}{\left(2\cos(\frac{j\pi}{p})\right)^m} \\
&= \frac{1}{p} + \frac{2}{p(k+1)^m} \cdot \sum_{r=0}^{(p-1)/2-1} \frac{\cos(\frac{(4q+1+4l)(\frac{p-1}{2}-r)\pi}{p})}{\left(2\cos(\frac{(\frac{p-1}{2}-r)\pi}{p})\right)^m}
\end{aligned}$$

The terrible-looking term in the last line can be simplified: notice that

$$\cos(\frac{(4q+1+4l)(\frac{p-1}{2}-r)\pi}{p}) = \cos\left(\frac{4q+4l+1}{2}\pi - \frac{(2r+1)(4q+4l+1)}{2p}\pi\right) = \sin(\frac{(2r+1)(4q+4l+1)}{2p}\pi)$$

and

$$\cos(\frac{(\frac{p-1}{2}-r)\pi}{p}) = \cos(\frac{\pi}{2} - \frac{2r+1}{2p}\pi) = \sin\left(\frac{2r+1}{2p}\pi\right)$$

Now we further simplify the probability to

$$\mathsf{Pr}\,[y=l] = \frac{1}{p} + \frac{2}{p(k+1)^m} \cdot \sum_{r=0}^{\frac{p-1}{2}-1} \frac{\sin\left(\frac{(2r+1)(4q+4l+1)}{2p}\pi\right)}{\left(2\sin\left(\frac{2r+1}{2p}\pi\right)\right)^m} \tag{20}$$

Now we define

$$D_r = \frac{\sin\left(\frac{(2r+1)(4q+4l+1)}{2p}\pi\right)}{\left(2\sin\left(\frac{2r+1}{2p}\pi\right)\right)^m}$$

and we will try to estimate D_r for $0.05p \le l \le 0.45p$.

Notice we have $p \ge 50m = 200q + 50$, we have

$$\frac{\pi}{10} < \frac{4q+4l+1}{2p}\pi < \frac{46\pi}{50}$$

and thus

$$\sin\left(\frac{4q+4l+1}{2p}\pi\right) > \sin(\frac{46\pi}{50}) > 0.248$$

Also we have

$$\sin\left(\frac{\pi}{2p}\right) < \frac{\pi}{2p}$$

So we have

$$D_0 = \frac{\sin\left(\frac{4q+4l+1}{2p}\pi\right)}{\left(2\sin\left(\frac{\pi}{2p}\right)\right)} > 0.248 \cdot (\frac{p}{\pi})^m \tag{21}$$

For the rest D_r's, we bound them from the other direction: first we always have:

$$\sin\left(\frac{(2r+1)(4q+4l+1)}{2p}\pi\right) > -1$$

and second, by expanding $\sin(x)$ to its Taylor Series, we have

$$\sin(x) > x - \frac{x^3}{3!} = x(1-\frac{x^2}{6})$$

and therefore we have

$$\sin\left(\frac{2r+1}{2p}\pi\right) > \frac{2r+1}{2p}\pi(1-\frac{(\pi/2)^2}{6}) \geq 1.849 \cdot \frac{2r+1}{2p}$$

since

$$0 < \frac{2r+1}{2p}\pi < \frac{\pi}{2}$$

Therefore we have

$$D_r = \frac{\sin\left(\frac{(2r+1)(4q+4l+1)}{2p}\pi\right)}{\left(2\sin\left(\frac{2r+1}{2p}\pi\right)\right)^m} \geq -\left(\frac{p}{1.849\cdot(2r+1)}\right)^m = -\left(\frac{p}{1.849}\right)^m \cdot \left(\frac{1}{2r+1}\right)^m \tag{22}$$

for $r > 1$.

Now combining the results from equation 21 and equation 22, we have:

$$\begin{aligned}\Pr[y=l] &= \frac{1}{p} + \frac{2}{p(k+1)^m} \cdot \sum_{r=0}^{\frac{p-1}{2}-1} D_r \\ &> \frac{1}{p} + \frac{2}{p(k+1)^m} \cdot \left(0.248 \cdot (\frac{p}{\pi})^m - \left(\frac{p}{1.849}\right)^m \cdot \sum_{r=1}^{\frac{p-1}{2}-1} \left(\frac{1}{2r+1}\right)^m\right)\end{aligned}$$

Now look at the function $g(r) = \left(\frac{1}{2r+1}\right)^m$. It is a convex function when $r > 0$, which means that

$$\int_{a-\frac{1}{2}}^{a+\frac{1}{2}} g(x)dx > g(a)$$

So

$$\sum_{r=1}^{\frac{p-1}{2}-1} \left(\frac{1}{2r+1}\right)^m < \int_{1/2}^{\infty} \left(\frac{1}{2r+1}\right)^m dr = \frac{1}{(m-1)\cdot 2^m} \tag{23}$$

Notice that

$$\frac{p}{(k+1)\pi} > 0.62, \quad \frac{p}{3.69(k+1)} < 0.5$$

Putting everything together:

$$\begin{aligned}\Pr[y=l] &> \frac{1}{p} + \frac{2}{p(k+1)^m} \cdot \left(0.248 \cdot (\frac{p}{\pi})^m - \left(\frac{p}{1.849}\right)^m \cdot \sum_{r=1}^{\frac{p-1}{2}-1} \left(\frac{1}{2r+1}\right)^m\right) \\ &> \frac{1}{p} + \frac{0.497}{p}\left(\frac{p}{(k+1)\pi}\right)^m - \frac{1}{2p(m-1)} \cdot \left(\frac{p}{3.69(k+1)}\right)^m \\ &> \frac{1}{p}(1+0.495 \cdot 0.62^m) \\ &> \frac{1}{p}(1+0.5^m)\end{aligned}$$

■

Lemma 19 shows a lower bound for the "good" elements in A, namely the elements between $0.05p$ and $0.45p$: they have a "marginal advantage" 0.5^m of being the sum of m elements from A.

The opposite of lemma 19 is also true, i.e., the "good" elements in B, namely the elements between $0.55p$ and $0.95p$ have a "marginal disadvantage" 0.5^m of being the sum of m elements from A:

Lemma 20 *Let m be a natural number such that $m \equiv 1 \pmod 4$ and $m > 10$. Let p be an odd prime number ($p \geq 2$) that is larger than $50m$. Let l be a natural number that that $0.55p < l < 0.95p$. Let A be the set $\{0, 1, ..., (p-1)/2\}$. Let $x_1, x_2, ..., x_m$ be random samples chosen uniformly from A and let $y = x_1 + x_2 + ...x_m \pmod p$. Then*

$$\mathsf{Pr}\left[y = l\right] < \frac{1}{p}(1 - 0.5^m)$$

The proof is essentially the same as the proof for lemma 19.

Now combining lemma 18, lemma 19 and lemma 20, we are ready to prove theorem 8:

Theorem 8 *With probability at least 0.8, the* BUILD-TREE *algorithm learns $\mathcal{F}$ with accuracy $1 - e^{-0.03n}$. In other words, the* BUILD-TREE *algorithm learns $\mathcal{F}$ with advantage 0.5, and has a running time $p^{O(n/\log n)}$.*

Proof: First BUILD-TREE aborts with very low probability and we don't have to worry about that.

We still denote the input vector by $\vec{X}$, and we use the same notations as in lemma 18. Notice the size of the root node is the number of $(\vec{u}_i, \vec{v}_i)$ pairs such that $\vec{u}_i + \vec{v}_i = \vec{0}$. By lemma 18 and theorem 19, we know that if $l = \vec{a} \cdot \vec{X}$ is within the range $[0.05p, 0.45p]$, then the probability that $\vec{u}_i + \vec{v}_i = \vec{0}$ is at least $(1 + 0.5^{2^a+1})/p^b$. if we define

$$\epsilon = 0.5^{2^a+1}$$

then the expected size of the root node is at least

$$\mu = K(1+\epsilon)/p^b = 2^{2^{a+1}} \cdot n(1+\epsilon)$$

By lemma 14, all these $(\vec{u}_i, \vec{v}_i)$ are independents, and thus by Chernoff bound, the probability that the actually size of the root is smaller than $2^{2^{a+1}} \cdot n$ is at most

$$e^{-(\frac{\epsilon}{1+\epsilon})^2 \cdot \mu/2} = e^{-(\frac{\epsilon}{1+\epsilon})^2 \cdot 2^{2^{a+1}} \cdot n(1+\epsilon)/2} \leq e^{-0.2 \cdot 0.5^{2^{a+1}+2} \cdot 2^{2^{a+1}} \cdot n} \leq e^{-0.05n}$$

So when $\vec{a} \cdot \vec{X} \in [0.05p, 0.45p]$, BUILD-TREE is correct with probability at least $1 - e^{-0.05n}$. Similarly, by lemma 18 and theorem 19, we know that when $\vec{a} \cdot \vec{X} \in [0.55p, 0.95p]$, the probability that $\vec{u}_i + \vec{v}_i = \vec{0}$ is at most $(1 - 0.5^{2^a+1})/p^b$, and again by Chernoff bound, the probability that the actual size of the root node is larger than $2^{2^{a+1}} \cdot n$ is at most

$$e^{-(\frac{\epsilon}{1+\epsilon})^2 \cdot \mu/3} = e^{-(\frac{\epsilon}{1+\epsilon})^2 \cdot 2^{2^{a+1}} \cdot n(1+\epsilon)/3} \leq e^{-0.15 \cdot 0.5^{2^{a+1}+2} \cdot 2^{2^{a+1}} \cdot n} \leq e^{-0.03n}$$

and BUILD-TREE is correct with probability at least $1 - e^{-0.03n}$.

When $\vec{a} \cdot \vec{X}$ is within the range $[0, 0.05p] \cup [0.45p, 0.55p] \cup [0.95p, p]$, BUILD-TREE might make a lot of mistakes, but that happens only with probability 0.2. So with probability 0.8, BUILD-TREE is correct with accuracy at least $1 - e^{-0.03n}$. By Lemma 1, BUILD-TREE has an advantage $0.6 - 2e^{-0.03n} > 0.5$.

Finally the running time: In the phase I of BUILD-TREE , $2^{a-1}(2^a+1)K = 2^{a-1}(2^a+1)2^{2^{a+1}}p^b$ negative samples are drawn, and that takes time $2^{O(a)} \cdot p^{O(n/\log n)} \cdot 2^{O(\sqrt{n})} = p^{O(n/\log n)}$. In the phase II, a binary tree of height a is constructed, where each node takes time at most $(2^{a-1}K)^2 = p^{O(n/\log n)}$, and there are $2^a - 1$ nodes to construct. So the total running time of BUILD-TREE is $p^{O(n/\log n)}$.

■

It is interesting to compare BUILD-TREE with the algorithm used in [BKW00], which also draws a lot of samples, view each sample as blocks, and try to write an input as the sum of $O(\log n)$ samples, and both algorithms have a similar sub-exponential bound. However, there are differences: in [BKW00], the algorithm draws samples with labels, and it writes an input as the sum of $O(\log n)$ samples to fight the noise — if there were no noise, it is easy to learn the function by Gauss elimination; in this paper, BUILD-TREE only draws negative examples, and it writes an input as the sum of $O(\log n)$ negative sample to create a probabilistic gap — there is no noise in the problem. Furthermore, the algorithm in [BKW00] is satisfied with justing finding a way to write an input as a sum of $O(\log n)$ samples, while BUILD-TREE has to estimate the probability that an input can be written as a sum of $O(\log n)$ samples, and thus is more complicated in this sense.

Notice that, using the same "padding" technique as in [BKW00], we can make the BUILD-TREE is polynomial-time algorithm: one simply pad $p^{n/\log n}$ zeros to the input of BUILD-TREE , and then BUILD-TREE 's running time becomes polynomial in the input length. However, still no polynomial-time algorithms can learn this class of linear threshold functions in statistical query model. This gives an example of PAC-learnable, but not SQ-learnable class of functions. Previously, both [K98] and [BFJ+94] proved that the class of parity functions fits into this category, and later [BKW00] proved that a class of noisy parity functions also fits. The linear threshold functions over a finite field is the first class of functions in this category that are not parity functions. We hope this result can provide further insights into SQ-learning algorithms.

5 Conclusions and Open Problems

In this paper, we discussed the problem of learning (possibly highly) correlated functions in the statistical query model. We showed an almost-tight upper bound of the advantage an algorithm can in approximating a class of functions simultaneously. We also show that any SQ algorithm trying to get a better advantage in learning the class of functions has to make a lot of queries. A consequence of our result is that the class of booleanized linear functions over finite fields are not SQ-learnable, which include linear threshold functions. Finally we demonstrated a PAC learning algorithm that learns a class of linear threshold functions with constant advantage and running time that is provably impossible for SQ-algorithms. With proper padding, our algorithm can be made in polynomial time, and thus putting linear threshold functions into the category of PAC-learnable, but not SQ-learnable functions, and they are the first class in this category that are not parity functions.

The technique we used in this paper to prove the lower bound is to keep track of the "all-pair statistical distance" between scenarios when the algorithm is given different target functions Our technique is similar to the one used in [A00], where the author proved a lower bound of quantum queries a quantum search algorithm has to make, but in a different setting. Their technique is to keep track of the sum of the absolute values of the off-diagonal entries in in the system's density matrix — we denote this quantity by S. Roughly speaking, the author in [A00] proved that:

1. Before the algorithm makes any quantum queries, S is large.
2. After the algorithm finishes all the queries, S is small.
3. Each quantum query only decreases S by a small amount.

And then they conclude that lot of quantum queries are needed. It would be interesting to investigate if there is a deeper relationship between the 2 techniques.

People already understand SQ-learning un-correlated functions: both lower bounds and upper bounds on the number of queries are shown, and the two bounds match. Our paper gives a lower bound for SQ-learing a class of functions that are correlated the same way, but no matching upper bound is known. Even less is known for the case that all the functions are correlated, but not in the same way. In general, given d functions $f_1, f_2, ..., f_d$ and their pair-wise correlation $\langle f_i, f_j \rangle$ for all $i \neq j$, can we find a good lower bound for the number of queries needed to learning these d functions well? Is there an (even non-uniform) matching upper bound?

Another interesting problem is: do there exist efficient algorithms to learn booleanized linear functions over finite fields? For parity functions over GF_2, they are easy to learn when there is no noise, and hard if there is noise — the state of art are Blum et al.'s algorithm [BKW00], which takes time $O(2^{n/\log n})$ for n-bit parity functions with respect to uniform noise of constant rate, and Goldreich-Levin-Jackson's algorithm [GL89, J00], which takes time $O(2^{n/2})$ for n-bit parity functions, with respect to uniform noise of rate $(1/2 - 1/\mathsf{poly}(n))$ and some classes of malicious noise. However, in the case of finite fields of large characteristics, it seems it is hard to learn the booleanized linear functions even without noise. Notice an efficient learning algorithm will break Baird's cryptosystem, and an hardness result will automatically translate to a security proof for Baird's system.

Another interesting topic is learning functions in finite fields in general: instead of limiting the outputs of functions to be boolean, we allow functions to output elements in a finite field, or some other large domains. What kind of functions are learnable?

Acknowledgement

Leemon Baird gave the initial motivation of this paper by mentioning his cryptosystem, which lead us into studying the problem of learning booleanized linear functions. Avrim Blum gave a lot of invaluable advice on what problems are interesting and helped formulate ideas into a paper. Adam Kalai and Steven Rudich helps clarify a lot of concepts and especially helped me understand what the right problems should be asked. Salil Vadhan showed me the idea of proving the sub-additivity of all-pair statistical distance, which is one of the essential tools used in the paper.

References

[A00] Andris Ambainis. *Quantum lower bounds by quantum arguments*, In *Proceedings of the 32nd ACM Symposium on Theory of Computing*, pages 636-643, 2000.

[AD98] Javed Aslam and Scott Decatur. *General Bounds on Statistical Query Learning and PAC learning with Noise via Hypothesis Boosting*, In *Information and Computation*, 141, pages 85-118 (1998).

[B01] Leemon Baird. *Blind Computation.* Manuscript, 2001.

[BFJ+94] Avrim Blum, Merrick Furst, Jeffrey Jackson, Michael Kearns, Yishay Mansour, and Steven Rudich. *Weakly Learning DNF and Characterizing Statistical Query Learning Using Fourier Analysis.* In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing*, pages 253–262, 1994.

[BFK+96] Avrim Blum, Alan Frieze, Ravi Kannan, and Santosh Vempala, *A Polynomial-time Algorithm for Learning Noisy Linear Threshold Functions*, In *Algorithmica, 22:35–52, 1998.* An extended abstract appears in *Proceedings of the 37th Annual Symposium on Foundations of Computer Science (FOCS'96)*, pages 330–338.

[BKW00] Avrim Blum, Adam Kalai and Hal Wasserman, *Noise-tolerant Learning, the Parity problem, and the Statistical Query model.* In *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing*, pp. 435–440, 2000.

[D95] Scott Decatur, *Efficient Learning from Faulty Data.* Ph.D. Thesis, Harvard University, TR-30-95, 1995.

[GL89] Oded Goldreich and Leonid Levin, *A hard-core predicate for all one-way functions.* In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing*, pp. 25-32, 1989.

[J00] Jeff Jackson *On the Efficiency of Noise-Tolerant PAC Algorithms Derived from Statistical Queries.* In *Proceedings of the 13th Annual Workshop on Computational Learning Theory*, 2000.

[K98] Michael Kearns. *Efficient noise-tolerant learning from statistical queries.* In *Journal of the ACM*, 45(6), pp. 983 — 1006, 1998. Preliminary version in *Proceedings of the 25th Annual ACM Symposium on Theory of Computing*, pp. 392–401, 1993.

[MR95] Rajeev Motwani and Prabhakar Raghavan, *Randomized Algorithms*, Cambridge University Press, 1995.

[SS96] Robert Schapire and Linda Selle, *Learning Sparse Multivariate Polynomials over a Field with Queries and Counterexamples.* In *Journal of Computer and System Sciences*, 52, 201-213, 1996.

[V01] Salil Vadhan, *Private Communication.*

[V84] Leslie Valiant, *A theory of the Leanable.* In *Communications of the ACM*, 27(11): 1134–1142, November 1984.