2006-03-22

Learning Real-World Problems by Finding Correlated Basis Functions

Adam C. Drake
Brigham Young University - Provo

LEARNING REAL-WORLD PROBLEMS BY FINDING CORRELATED BASIS FUNCTIONS

by

Adam Drake

A thesis submitted to the faculty of

Brigham Young University

in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computer Science

Brigham Young University

April 2006

BRIGHAM YOUNG UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a thesis submitted by

Adam Drake

This thesis has been read by each member of the following graduate committee and by majority vote has been found to be satisfactory.

Date	Dan Ventura, Chair
Date	Tony R. Martinez
Date	Irene Langkilde-Geary

BRIGHAM YOUNG UNIVERSITY

As chair of the candidate's graduate committee, I have read the thesis of Adam Drake in its final form and have found that (1) its format, citations, and bibliographical style are consistent and acceptable and fulfill university and department style requirements; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the graduate committee and is ready for submission to the university library.

Date

Dan Ventura
Chair, Graduate Committee

Accepted for the Department

Parris K. Egbert
Graduate Coordinator

Accepted for the College

Thomas W. Sederberg
Associate Dean, College of Physical and
Mathematical Sciences

ABSTRACT

LEARNING REAL-WORLD PROBLEMS BY FINDING CORRELATED BASIS FUNCTIONS

Adam Drake

Department of Computer Science

Master of Science

Learning algorithms based on the Fourier transform attempt to learn functions by approximating the largest coefficients of their Fourier representations. Nearly all previous work in Fourier-based learning has been in the theoretical realm, where properties of the transform have made it possible to prove many interesting learnability results. The real-world usefulness of Fourier-based methods, however, has not been thoroughly explored. This thesis explores methods for the practical application of Fourier-based learning.

The primary contribution of this thesis is a new search algorithm for finding the largest coefficients of a function's Fourier representation. Although the search space is exponentially large, empirical results demonstrate that only a small fraction of the space needs to be explored to find the largest coefficients. Furthermore, the algorithm is applicable to a much wider range of learning scenarios than previous approaches.

Results of learning real-world problems with algorithms based on this search technique are also presented. The accuracies of the Fourier-based learning methods

are not particularly impressive, however, and analysis and empirical results suggest why the Fourier representation may be a poor choice for typical real-world learning problems.

Finally, this thesis shows that the search algorithm can be generalized to explore any basis of functions. Furthermore, it can search multiple bases simultaneously. This greatly enhances the learning techniques, and empirical results demonstrate significantly improved accuracy over the Fourier-based approach.

Contents

Chapter 1

Introduction

The process of learning by example can be thought of as the process of approximating a function whose output is only partially known. Each known point of the function is one training example. From the known points of the function, a learning algorithm derives an approximation of the unknown function.

An interesting twist on the function approximation perspective of machine learning was introduced in 1989 by Linial, Mansour, and Nisan [17]. They proposed a learning method in which a function is approximated indirectly by approximating its Fourier representation. In its Fourier representation, a function is represented as a linear combination of Fourier basis functions.

The initial work of Linial, Mansour, and Nisan, as well as nearly all Fourier-based learning work since, has been in the area of learning theory. The mathematical properties of the Fourier transform have made it possible to prove many interesting learnability results, making it one of the most useful tools in computational learning theory.

Despite successes in the theoretical realm, however, there has been little exploration into the practicality of using Fourier-based learning techniques to solve real-world problems. This thesis explores the real-world potential of Fourier-based learning, presenting both positive and negative results.

One of the primary contributions of this thesis is a new technique for computing Fourier spectra. When integrated into a best-first search, it is possible to very quickly determine which of the exponential number of Fourier basis functions

are most highly correlated with available training data, allowing the Fourier representation to be approximated efficiently. This best-first search algorithm overcomes several limitations of previous methods, and although it does not have the polynomial run-time guarantees of some previous approaches, it has been found to be very efficient in practice.

Building on these spectral computation methods, this thesis presents results of learning real-world problems with Fourier-based learning algorithms. Experiments demonstrate that Fourier-based learning algorithms can solve real-world learning problems with a high degree of accuracy, although the accuracies achieved are generally lower than those achieved by existing algorithms.

The mediocre learning results of the Fourier-based algorithms exemplify an interesting negative result presented in this thesis. In analyzing the Fourier representation, arguments and empirical results are presented that suggest that learning algorithms that are based on XOR relationships, such as the Fourier-based algorithms, are less well-suited to real-world learning problems than algorithms based on logical AND and OR relationships. Consequently, although Fourier-based algorithms are useful for proving theoretical results, they may be less useful for solving real-world learning problems.

Although the conclusion that Fourier-based learning algorithms may not be well-suited to typical real-world machine learning problems is negative, its discovery leads to another positive result. The best-first search technique can be generalized to search for any type of function. Thus, the learning techniques can also be performed with sets of AND and OR functions, which appear to be more useful for solving real-world problems. The addition of AND and OR functions results in significantly improved learning accuracy.

The remainder of this thesis is organized as follows: Chapter 2 gives the necessary background information on the Fourier transform of Boolean functions and describes previous work in Fourier-based learning. Chapter 3 describes the new technique for computing Fourier spectra. Chapter 4 presents results of applying Fourier-based learning algorithms to real-world learning problems and contains an analysis

of the Fourier representation. Chapter 5 describes how the Fourier techniques can be generalized, allowing the Fourier basis functions to be replaced or supplemented with other functions. Finally, Chapter 6 contains concluding remarks and suggests directions for future research.

(Portions of this thesis have appeared in two other publications. Parts of Chapter 4 appeared in the *Proceedings of the 8th Joint Conference on Information Sciences* [5], and parts of Chapters 3, 4, and 5 appeared in the *Proceedings of the 22nd International Conference on Machine Learning* [6].)

Chapter 2

Background

This chapter provides the background information upon which the remainder of the thesis is built. It describes the Fourier transform of functions of Boolean inputs, introduces the notation that will be used throughout the thesis, and summarizes previous work in Fourier-based learning. Finally, some motivation for the work of this thesis is provided.

2.1 Preliminaries

This thesis is concerned with functions of one or more Boolean inputs and a single Boolean output ($f : \{0,1\}^n \longrightarrow \{1,-1\}$). For convenience in performing Fourier analysis, inputs are encoded as 0s and 1s, while outputs are encoded as 1s and -1s. By convention, 1 denotes a negative output, while -1 denotes a positive output.

As an example of such a Boolean function, consider the function f_{OR} that computes the logical OR of two inputs, x_0 and x_1, as shown in the following truth table:

x_0	x_1	$f_{OR}(x)$
0	0	1
0	1	-1
1	0	-1
1	1	-1

Notice that x represents the vector of all inputs to f, while x_i represents the i^{th} input.

There are 2^n possible input combinations for a function of n Boolean inputs. Consequently, a Boolean function of n inputs can be defined by a vector of length 2^n, in which each entry gives the output of one input combination. By converting each set of Boolean inputs into an n-digit binary number, the binary number ordering provides a natural ordering for indexing the outputs. The function f_{OR} described above is represented in vector form as follows:

$$f_{OR} = \begin{bmatrix} 1 \\ -1 \\ -1 \\ -1 \end{bmatrix}$$

2.2 The Fourier Transform

The Fourier transform used in this thesis is a simplified version of the discrete Fourier transform that applies to functions of Boolean inputs. It is also known as a Walsh transform, although in this thesis it will generally be referred to simply as the Fourier transform.

The Fourier transform will be defined here in two ways. It will first be defined using the function notation commonly used in Fourier-based learning literature. It will then be defined in terms of Walsh matrices. The two definitions are equivalent, although the matrix definition may be easier to visualize.

2.2.1 Function Definition

Before showing how a function is transformed into its Fourier representation, it will be necessary to define the Fourier basis functions.

The Fourier basis functions are defined by the following:

$$\chi_\alpha(x) = (-1)^{\sum_{i=0}^{n-1} \alpha_i x_i} \tag{2.1}$$

where $\alpha, x \in \{0,1\}^n$, and α_i and x_i are the i^{th} digits of α and x. The binary number α is the label of the basis function, and uniquely identifies it. The following example

shows how the 2-input basis function χ_{10} is derived from the previous definition:

$$\chi_{10} = \begin{bmatrix} \chi_{10}(00) \\ \chi_{10}(01) \\ \chi_{10}(10) \\ \chi_{10}(11) \end{bmatrix} = \begin{bmatrix} (-1)^{(1\cdot0)+(0\cdot0)} \\ (-1)^{(1\cdot0)+(0\cdot1)} \\ (-1)^{(1\cdot1)+(0\cdot0)} \\ (-1)^{(1\cdot1)+(0\cdot1)} \end{bmatrix} = \begin{bmatrix} (-1)^0 \\ (-1)^0 \\ (-1)^1 \\ (-1)^1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \end{bmatrix}$$

The Fourier basis functions are parity functions, each computing the parity (or logical XOR) of a subset of inputs. This is more easily seen in the following definition:

$$\chi_\alpha(x) = \begin{cases} 1 : \text{if } \sum_{i=0}^{n-1} \alpha_i x_i \text{ is even} \\ -1 : \text{if } \sum_{i=0}^{n-1} \alpha_i x_i \text{ is odd} \end{cases} \tag{2.2}$$

The subset of inputs over which parity is computed is determined by the basis function's label, α. Notice that for each i such that $\alpha_i = 0$ the result of the sum is unchanged because $0 \cdot x_i = 0$ regardless of the value of x_i. Consequently, the output of each basis function is determined by the parity of the inputs for which $\alpha_i = 1$.

Because the basis functions compute the parity of a subset of inputs, it is natural to define them in terms of those subsets. Let $S \subseteq \{0, 1, ..., n-1\}$ be the set of all i such that $\alpha_i = 1$. Then the Fourier basis functions can also be defined as follows:

$$\chi_S(x) = \begin{cases} 1 : \text{if } \sum_{i \in S} x_i \text{ is even} \\ -1 : \text{if } \sum_{i \in S} x_i \text{ is odd} \end{cases} \tag{2.3}$$

Having defined the basis functions, it is now possible to define the Fourier transform. Let f be a Boolean function of the form $f : \{0,1\}^n \longrightarrow \{1,-1\}$. Then the Fourier representation, or Fourier spectrum, of f, signified by $\hat{f}$, is given by the following:

$$\hat{f}(\alpha) = \frac{1}{2^n} \sum_{x \in \{0,1\}^n} f(x)\chi_\alpha(x) \tag{2.4}$$

in which $\alpha \in \{0,1\}^n$. Each $\hat{f}(\alpha)$ is one of the Fourier coefficients of $\hat{f}$. Notice that each Fourier coefficient is the normalized dot product of f and one of the basis functions.

The following example shows how Equation 2.4 can be used to compute Fourier spectra. Let f be a function of two inputs defined by the following vector:

$$f = \begin{bmatrix} 1 \\ -1 \\ -1 \\ -1 \end{bmatrix}$$

Then $\hat{f}$ is computed as follows:

$$\begin{aligned}
\hat{f} &= \begin{bmatrix} \hat{f}(00) \\ \hat{f}(01) \\ \hat{f}(10) \\ \hat{f}(11) \end{bmatrix} \\
&= \begin{bmatrix} \frac{1}{4}\sum_{x\in\{0,1\}^2} f(x)\chi_{00}(x) \\ \frac{1}{4}\sum_{x\in\{0,1\}^2} f(x)\chi_{01}(x) \\ \frac{1}{4}\sum_{x\in\{0,1\}^2} f(x)\chi_{10}(x) \\ \frac{1}{4}\sum_{x\in\{0,1\}^2} f(x)\chi_{11}(x) \end{bmatrix} \\
&= \begin{bmatrix} \frac{1}{4}(f(00)\chi_{00}(00) + f(01)\chi_{00}(01) + f(10)\chi_{00}(10) + f(11)\chi_{00}(11)) \\ \frac{1}{4}(f(00)\chi_{01}(00) + f(01)\chi_{01}(01) + f(10)\chi_{01}(10) + f(11)\chi_{01}(11)) \\ \frac{1}{4}(f(00)\chi_{10}(00) + f(01)\chi_{10}(01) + f(10)\chi_{10}(10) + f(11)\chi_{10}(11)) \\ \frac{1}{4}(f(00)\chi_{11}(00) + f(01)\chi_{11}(01) + f(10)\chi_{11}(10) + f(11)\chi_{11}(11)) \end{bmatrix} \\
&= \begin{bmatrix} \frac{1}{4}((1\cdot 1) + (-1\cdot 1) + (-1\cdot 1) + (-1\cdot 1)) \\ \frac{1}{4}((1\cdot 1) + (-1\cdot -1) + (-1\cdot 1) + (-1\cdot -1)) \\ \frac{1}{4}((1\cdot 1) + (-1\cdot 1) + (-1\cdot -1) + (-1\cdot -1)) \\ \frac{1}{4}((1\cdot 1) + (-1\cdot -1) + (-1\cdot -1) + (-1\cdot 1)) \end{bmatrix} \\
&= \begin{bmatrix} -0.5 \\ 0.5 \\ 0.5 \\ 0.5 \end{bmatrix}
\end{aligned}$$

Given $\hat{f}$, f can be retrieved by applying the inverse transform:

$$f(x) = \sum_{\alpha \in \{0,1\}^n} \hat{f}(\alpha)\chi_\alpha(x) \tag{2.5}$$

The following example uses Equation 2.5 to retrieve the function used in the previous example:

$$\begin{aligned}
f &= \begin{bmatrix} f(00) \\ f(01) \\ f(10) \\ f(11) \end{bmatrix} \\
&= \begin{bmatrix} \sum_{\alpha \in \{0,1\}^2} \hat{f}(\alpha)\chi_\alpha(00) \\ \sum_{\alpha \in \{0,1\}^2} \hat{f}(\alpha)\chi_\alpha(01) \\ \sum_{\alpha \in \{0,1\}^2} \hat{f}(\alpha)\chi_\alpha(10) \\ \sum_{\alpha \in \{0,1\}^2} \hat{f}(\alpha)\chi_\alpha(11) \end{bmatrix} \\
&= \begin{bmatrix} \hat{f}(00)\chi_{00}(00) + \hat{f}(01)\chi_{01}(00) + \hat{f}(10)\chi_{10}(00) + \hat{f}(11)\chi_{11}(00) \\ \hat{f}(00)\chi_{00}(01) + \hat{f}(01)\chi_{01}(01) + \hat{f}(10)\chi_{10}(01) + \hat{f}(11)\chi_{11}(01) \\ \hat{f}(00)\chi_{00}(10) + \hat{f}(01)\chi_{01}(10) + \hat{f}(10)\chi_{10}(10) + \hat{f}(11)\chi_{11}(10) \\ \hat{f}(00)\chi_{00}(11) + \hat{f}(01)\chi_{01}(11) + \hat{f}(10)\chi_{10}(11) + \hat{f}(11)\chi_{11}(11) \end{bmatrix} \\
&= \begin{bmatrix} (-0.5 \cdot 1) + (0.5 \cdot 1) + (0.5 \cdot 1) + (0.5 \cdot 1) \\ (-0.5 \cdot 1) + (0.5 \cdot -1) + (0.5 \cdot 1) + (0.5 \cdot -1) \\ (-0.5 \cdot 1) + (0.5 \cdot 1) + (0.5 \cdot -1) + (0.5 \cdot -1) \\ (-0.5 \cdot 1) + (0.5 \cdot -1) + (0.5 \cdot -1) + (0.5 \cdot 1) \end{bmatrix} \\
&= \begin{bmatrix} 1 \\ -1 \\ -1 \\ -1 \end{bmatrix}
\end{aligned}$$

2.2.2 Matrix Definition

The Fourier transform can also be represented by a matrix in which each row is the vector of outputs of one of the Fourier basis functions. For a function of n

inputs, the matrix is a $2^n \times 2^n$ matrix in which each element W_{ij} is given by:

$$W_{ij} = (-1)^{\sum_{k=0}^{n-1} i_k j_k} \tag{2.6}$$

where i_k and j_k are the k^{th} digits of the binary representations of indices i and j. (Note that this formula is equivalent to that given in Equation 2.1.) This matrix is commonly known as a Walsh or Hadamard matrix. The Walsh transform matrices for functions of one and two inputs, W^1 and W^2, are shown here, with each row labeled by the basis function it represents:

$$W^1 = \begin{matrix} \chi_0 \\ \chi_1 \end{matrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \qquad W^2 = \begin{matrix} \chi_{00} \\ \chi_{01} \\ \chi_{10} \\ \chi_{11} \end{matrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$

W^1 and W^2 are derived from Equation 2.6 as follows:

$$\begin{aligned} W^1 &= \begin{bmatrix} (-1)^{0\cdot 0} & (-1)^{0\cdot 1} \\ (-1)^{1\cdot 0} & (-1)^{1\cdot 1} \end{bmatrix} \\ &= \begin{bmatrix} (-1)^0 & (-1)^0 \\ (-1)^0 & (-1)^1 \end{bmatrix} \\ &= \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \end{aligned}$$

$$
\begin{aligned}
W^2 &= \begin{bmatrix}
(-1)^{0\cdot0+0\cdot0} & (-1)^{0\cdot0+0\cdot1} & (-1)^{0\cdot1+0\cdot0} & (-1)^{0\cdot1+0\cdot1} \\
(-1)^{0\cdot0+1\cdot0} & (-1)^{0\cdot0+1\cdot1} & (-1)^{0\cdot1+1\cdot0} & (-1)^{0\cdot1+1\cdot1} \\
(-1)^{1\cdot0+0\cdot0} & (-1)^{1\cdot0+0\cdot1} & (-1)^{1\cdot1+0\cdot0} & (-1)^{1\cdot1+0\cdot1} \\
(-1)^{1\cdot0+1\cdot0} & (-1)^{1\cdot0+1\cdot1} & (-1)^{1\cdot1+1\cdot0} & (-1)^{1\cdot1+1\cdot1}
\end{bmatrix} \\
&= \begin{bmatrix}
(-1)^0 & (-1)^0 & (-1)^0 & (-1)^0 \\
(-1)^0 & (-1)^1 & (-1)^0 & (-1)^1 \\
(-1)^0 & (-1)^0 & (-1)^1 & (-1)^1 \\
(-1)^0 & (-1)^1 & (-1)^1 & (-1)^2
\end{bmatrix} \\
&= \begin{bmatrix}
1 & 1 & 1 & 1 \\
1 & -1 & 1 & -1 \\
1 & 1 & -1 & -1 \\
1 & -1 & -1 & 1
\end{bmatrix}
\end{aligned}
$$

The Walsh-Hadamard matrices can also be defined recursively. Let W^0 be the basis case:

$$W^0 = 1 \tag{2.7}$$

Then for all $n > 0$, W^n can be defined as follows:

$$W^n = \begin{bmatrix} W^{n-1} & W^{n-1} \\ W^{n-1} & -W^{n-1} \end{bmatrix} \tag{2.8}$$

This recursive pattern can be observed in W^1 and W^2, shown previously. For example, notice that each quadrant of W^2 is W^1, with the lower-right quadrant inverted.

The Walsh matrices can be used to compute the Fourier representation $\hat{f}$ of a function f by taking the product of f and the appropriate Walsh matrix, and then normalizing the result:

$$\hat{f} = \frac{1}{2^n} W^n \cdot f \tag{2.9}$$

For example, let f be the two-input function used in Section 2.2.1, shown again here:

$$f = \begin{bmatrix} 1 \\ -1 \\ -1 \\ -1 \end{bmatrix}$$

The Fourier representation of f is computed as follows:

$$
\begin{aligned}
\hat{f} &= \frac{1}{2^2} W^2 \cdot f \\
&= \frac{1}{4} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ -1 \\ -1 \\ -1 \end{bmatrix} \\
&= \frac{1}{4} \begin{bmatrix} (1 \cdot 1) + (1 \cdot -1) + (1 \cdot -1) + (1 \cdot -1) \\ (1 \cdot 1) + (-1 \cdot -1) + (1 \cdot -1) + (-1 \cdot -1) \\ (1 \cdot 1) + (1 \cdot -1) + (-1 \cdot -1) + (-1 \cdot -1) \\ (1 \cdot 1) + (-1 \cdot -1) + (-1 \cdot -1) + (1 \cdot -1) \end{bmatrix} \\
&= \frac{1}{4} \begin{bmatrix} -2 \\ 2 \\ 2 \\ 2 \end{bmatrix} \\
&= \begin{bmatrix} -0.5 \\ 0.5 \\ 0.5 \\ 0.5 \end{bmatrix}
\end{aligned}
$$

The function f can be retrieved from $\hat{f}$ by applying the same transform to $\hat{f}$, without the normalization:

$$f = W^n \cdot \hat{f} \tag{2.10}$$

The following example demonstrates this:

$$
\begin{aligned}
f &= W^2 \cdot \hat{f} \\
&= \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \cdot \begin{bmatrix} -0.5 \\ 0.5 \\ 0.5 \\ 0.5 \end{bmatrix} \\
&= \begin{bmatrix} (1 \cdot -0.5) + (1 \cdot 0.5) + (1 \cdot 0.5) + (1 \cdot 0.5) \\ (1 \cdot -0.5) + (-1 \cdot 0.5) + (1 \cdot 0.5) + (-1 \cdot 0.5) \\ (1 \cdot -0.5) + (1 \cdot 0.5) + (-1 \cdot 0.5) + (-1 \cdot 0.5) \\ (1 \cdot -0.5) + (-1 \cdot 0.5) + (-1 \cdot 0.5) + (1 \cdot 0.5) \end{bmatrix} \\
&= \begin{bmatrix} 1 \\ -1 \\ -1 \\ -1 \end{bmatrix}
\end{aligned}
$$

See [10] for a more detailed discussion of these and other aspects of Boolean spectral analysis.

2.2.3 Properties of the Fourier Transform

As mentioned in Section 2.2.1, the Fourier basis functions compute the XOR of a subset of their inputs. There is one basis function for every possible subset of inputs, including the empty set. The cardinality of the subset determines the order of the function. The basis function corresponding to the empty set is the 0^{th}-order basis function. It computes the XOR of no inputs, and therefore has constant output. For all $n > 0$, there are n 1^{st}-order basis functions whose outputs are determined by a single input. These functions output 1 if that input is 0, and output -1 otherwise. The remaining higher-order basis functions compute the XOR of 2 or more inputs.

Another important property of the Fourier transform is that the Fourier basis functions form an orthonormal basis. This means that the Fourier basis functions are

orthogonal to each other, and that every Boolean function can be represented as a linear combination of Fourier basis functions.

Finally, the Fourier coefficients show how well correlated a function is with the basis functions. Recall that each Fourier coefficient is computed by taking the dot product of f and some basis function χ_α. The dot product reveals the level of correlation between two vectors. Consequently, a large positive or negative value for $\hat{f}(\alpha)$ indicates a strong positive or negative correlation between f and χ_α. A value of 0 for $\hat{f}(\alpha)$ indicates that there is no correlation between f and χ_α.

The distinction between positive and negative correlation is generally not important here. Therefore, in this thesis, a "large" coefficient will refer to a coefficient with a large absolute value. Similarly, a basis function will be referred to as being "highly correlated" if its coefficient is large, regardless of whether it is a positive or negative correlation.

2.3 Fourier-based Learning

This section shows how the Fourier transform can be used to learn Boolean functions. It also describes previous work in this area.

2.3.1 Learning Fourier Representations

Given examples of an unknown function, Fourier-based algorithms attempt to learn the function by learning the coefficients of its Fourier representation. Because there are an exponential number of basis functions for a function of n inputs, it is generally not feasible to approximate the entire Fourier representation. Therefore, Fourier-based learning algorithms generally approximate the Fourier coefficients of only a subset of the basis functions. The task of learning Fourier representations can therefore be looked at as a two part process: determining which basis functions to use and determining what coefficient values to assign to them.

Previous approaches for determining which basis functions to use are presented in detail in the following section. The basic idea behind all approaches is to select

basis functions that appear to be highly correlated with the target function. These basis functions will have the largest coefficients.

Although the true coefficients cannot be computed without observing the value of the function at all points, they can be approximated by using only the available training examples.[1] Let X be the set of all x for which $f(x)$ is known. Then the Fourier coefficients can be approximated by the following:

$$\tilde{\hat{f}}(\alpha) = \frac{1}{|X|} \sum_{x \in X} f(x)\chi_\alpha(x) \tag{2.11}$$

The only changes made to the previous definition of Fourier coefficients (Equation 2.4) are that the sum is now computed over only the known points of f, rather than over all points, and the result is now divided by the number of known points, rather than by 2^n.

If unknown values of f are set to 0, the matrix definition (Equation 2.9) can be used to approximate the coefficients by changing the normalization factor from $\frac{1}{2^n}$ to $\frac{1}{|X|}$:

$$\tilde{\hat{f}} = \frac{1}{|X|} W^n \cdot f \tag{2.12}$$

After approximating the coefficients of the selected basis functions, a Fourier-based learning algorithm uses the linear combination of basis functions as its approximation, or hypothesis, of the target function f. Let A be the set of basis functions selected by a Fourier-based learning algorithm. Then the hypothesis $\tilde{f}$ produced by the algorithm is represented as follows:

$$\tilde{f}(x) = \sum_{\alpha \in A} \tilde{\hat{f}}(\alpha)\chi_\alpha(x) \tag{2.13}$$

The following example illustrates a simple Fourier-based learning method that selects the basis function with the largest coefficient to use as its hypothesis. Recall that the basis function with the largest coefficient will be the function that is most highly correlated with the training examples.

[1] For simplicity, throughout the remainder of this thesis, all references made to computing Fourier coefficients will refer to computing approximate coefficients as defined in Equations 2.11 and 2.12.

Suppose that the algorithm is used to learn a function of two inputs, for which three examples are known:

$$f = \begin{bmatrix} 1 \\ 0 \\ -1 \\ 1 \end{bmatrix}$$

After approximating the coefficients according to Equation 2.11 or 2.12, the last coefficient, $\hat{f}(11)$, is revealed to be the largest:

$$\tilde{\hat{f}} = \begin{bmatrix} .33 \\ -.33 \\ .33 \\ 1 \end{bmatrix}$$

Because $\hat{f}(11)$ is the largest coefficient, basis function χ_{11} will be the hypothesis:

$$\tilde{f} = \chi_{11} = \begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \end{bmatrix}$$

Using this hypothesis, the unknown output of the function will be assigned a value of -1.

The previous example of Fourier-based learning used a single basis function as the hypothesis, allowing the output of the hypothesis to be determined by the output of the selected basis function. More commonly, the hypothesis is a linear combination of Fourier basis functions and the output is determined by the sign of the linear combination. Let c be a Fourier-based classifier, and let A be the set of basis functions selected by the learner. Then the output of c for any input x is given by the following:

$$c(x) = sgn\left(\sum_{\alpha \in A} \hat{f}(\alpha)\chi_\alpha(x)\right)$$

where sgn is defined as follows:

$$sgn(x) = \begin{cases} 1 : \text{if } x \geq 0 \\ -1 : \text{if } x < 0 \end{cases}$$

2.3.2 Previous Work

Two significant Fourier-based learning algorithms, referred to here as the LMN and KM algorithms, have been introduced in the field of computational learning theory. Both have been used to prove bounds on the time, number of examples, etc., required to learn certain classes of functions under certain conditions. This section describes these algorithms, as well as a significant extension of the KM algorithm, referred to as the HS algorithm.

The LMN Algorithm

The LMN algorithm of Linial, Mansour, and Nisan was the first Fourier-based learning algorithm introduced [18]. Their key observation was that the Fourier representations of AC^0 functions (functions computable by constant-depth circuits of alternating levels of AND and OR gates) are almost entirely concentrated on the low-order coefficients. In other words, the coefficients of the high-order basis functions are either zero or are close enough to zero that their influence is negligible.

Because the Fourier spectra of AC^0 functions are concentrated on the low-order coefficients, it is possible to approximate AC^0 functions very well by approximating only the low-order coefficients. This idea forms the basis of the LMN algorithm. The LMN algorithm uses the training examples to learn the approximate values of the low-order coefficients. Using this algorithm, they proved that AC^0 functions can be learned in $O(n^{polylog(n)})$ time (assuming the presence of sufficient training examples drawn from a uniform distribution).

The KM Algorithm

The second major Fourier-based learning algorithm is the KM algorithm of Kushilevitz and Mansour [16]. Their algorithm, which is based on a technique of

Goldreich and Levin [9], searches the Fourier spectrum to find, with high probability, the largest coefficients. They prove that any function that can be approximated well by a function whose Fourier representation contains only a polynomial number of non-zero Fourier coefficients can be learned well by their algorithm in polynomial time. An interesting set of such functions is the set of functions representable by decision trees with Fourier basis functions at the nodes.

While the LMN algorithm is limited to learning functions whose large Fourier coefficients reside on the low-order basis functions, the KM algorithm can find large coefficients in any region of the Fourier spectrum (provided there are not too many). However, the KM algorithm has the disadvantage that it requires a membership oracle. (A membership oracle allows a learning algorithm to ask for the value of the unknown function at any point; in other words, during its training phase, the algorithm can select the training examples it will use.) The requirement of a membership oracle limits real-world applicability.

Although the requirement of a membership oracle limits its applicability, Mansour and Sahar provided a successful example of learning a real-world problem with the KM algorithm [20]. Their particular application involved reverse-engineering a state-free controller. Since the chip could be queried on any inputs, the chip could serve as the membership oracle. Unfortunately, however, access to a membership oracle is either impractical or impossible in many learning scenarios.

The HS Algorithm

The HS (Harmonic Sieve) algorithm of Jackson [11] combines the KM algorithm's search technique with a boosting algorithm of Freund [7]. The algorithm begins by using the KM algorithm to find one basis function that is highly correlated with the original distribution of training examples. Then, the distribution is updated to give more weight to examples misclassified by the previously selected basis function, and a new basis function is found based on the new distribution. This step is repeated until enough basis functions are found, after which the final hypothesis becomes a linear combination of the selected basis functions, with the weights of the

linear combination provided by the boosting algorithm.

Using the HS algorithm, Jackson was able to prove that DNF expressions could be learned in polynomial time with membership queries.

Other Fourier-Based Work

Since their initial creation, the previously described Fourier-based algorithms have been improved and used to prove many new learnability results.

Several results are based on the idea introduced in the LMN algorithm of approximating the low-order coefficients. For example, Bshouty and Tamon showed that monotone Boolean functions could be learned in this way since the Fourier spectra of monotone functions are concentrated on the low-order coefficients [4]. Klivans, O'Donnell, and Servedio proved a similar result for intersections and thresholds of halfspaces (linear thresholds) [14]. Jackson, Klivans, and Servedio combined the "low-order" Fourier method with a boosting algorithm to show that a more expressive class of functions than AC^0 can be learned in the time required by the LMN algorithm to learn AC^0 [12].

Prior to the HS algorithm and its ability to learn DNF expressions in polynomial time, others had used the KM algorithm to demonstrate weaker learnability results for DNF expressions. Mansour showed that the KM algorithm could be used to learn DNF in $O(n^{\log \log n})$ time [19]. Blum et. al. showed that it could be used to weakly learn DNF expressions in polynomial time [1]. Meanwhile, Khardon showed that various subsets of DNF could be learned by the KM algorithm [13].

Improvements to the HS algorithm have been presented by Klivans and Servedio [15] and Bshouty, Jackson, and Tamon [3]. In both cases, improved methods for finding correlated parity functions reduced the algorithm's complexity significantly.

Another Fourier-based approach was introduced by Bshouty and Feldman, who altered the KM and HS algorithms to use statistical queries rather than membership queries (they can query the oracle for probabilistic information about the target function instead of for labeled examples) [2]. Their approach is also similar to the low-order approaches in that it bounds the maximum order of considered basis functions.

2.4 Motivation

The motivation for this thesis is to determine if Fourier-based learning algorithms can be useful for solving real-world problems. Although Fourier-based algorithms have been extremely useful in proving theoretical learnability results, the assumptions and conditions required by those algorithms do not generally apply to real-world learning scenarios.

Because one of the goals of theoretical machine learning is to determine what can be learned efficiently, proven run-time and learnability bounds are critically important. Unfortunately, however, those results are specific to certain classes of functions. The theoretical learning algorithms are designed to learn functions in those classes, relying on assumptions that do not hold in general.

For example, consider the LMN algorithm. It is able to avoid exploring the entire Fourier spectrum because the spectral representations of AC^0 functions are concentrated on the low-order coefficients. However, there is no guarantee that the target function of an arbitrary real-world learning problem can be represented well by approximating only the low-order coefficients.

Another reason why theoretical Fourier-based algorithms may not make effective or practical real-world algorithms is that they are restricted to specific learning scenarios. For example, the KM algorithm requires membership queries to be able to carry out its search efficiently, and requires a uniform distribution to guarantee its learnability results. The problem with such restrictive assumptions is that they are often not typical of real-world scenarios. For example, membership queries are often not possible. Also, the true distribution of examples is often unknown, and is typically very non-uniform.

In short, the techniques used to produce Fourier-based theoretical results are not always helpful for solving real-world problems. The real-world learning scenario considered in this thesis is one in which the training set is fixed (no queries are allowed), the distribution of training examples is unknown, and nothing is known *a priori* about the target function (and therefore nothing is known about its Fourier representation).

The challenge is to create a Fourier-based learning algorithm that is effective under these conditions and runs in an acceptable amount of time. The following chapter presents a successful result in this research direction — a practical search algorithm for finding the basis functions that are most highly correlated with an arbitrary set of data.

Chapter 3

A New Algorithm for Finding Correlated Basis Functions

Ideally, a real-world Fourier-based learning algorithm would be capable of finding all Fourier basis functions that are highly correlated with an arbitrary set of training data, regardless of which basis functions those are. Unfortunately, for this general problem, there is no known polynomial-time solution.

And while no polynomial-time solution is presented here, this chapter does present an efficient best-first search algorithm that demonstrates that the basis functions that are most highly correlated with real-world data sets can be found quickly. Although the algorithm's worst-case time complexity is exponential, experiments on real-world problems suggest that in practice the algorithm can perform very well, needing to explore only a fraction of the search space to find the most highly correlated basis functions.

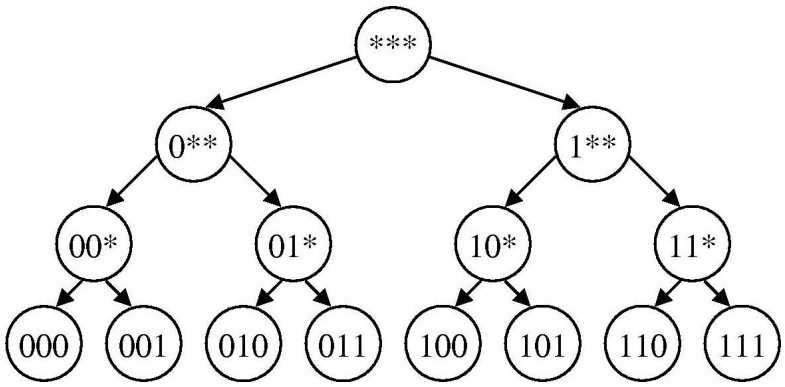
3.1 The Algorithm

The best-first search algorithm for finding correlated basis functions capitalizes on the fact that upper bounds on the largest possible coefficients in any region of the Fourier spectrum can be efficiently computed from available training data. Given these bounds, regions of the Fourier spectrum can be explored in order of largest possible coefficient, making it possible to find the basis functions with the largest coefficients while ignoring large portions of the search space.

3.1.1 The Search Space

The search space of the algorithm is the space of all partially and completely defined basis function labels. It can be viewed as a binary tree of nodes (see Figure 3.1).

Figure 3.1: A visualization of the search tree of the best-first search algorithm when $n = 3$. Internal nodes represent partially defined basis function labels, while each leaf node represents the label of a specific basis function.



Each basis function is identified by an n-digit binary label $\alpha \in \{0, 1\}^n$. There are 2^n unique α, corresponding to the 2^n basis functions. These α are the leaf nodes of the search tree.

Intermediate nodes in the search tree represent partially defined basis function labels. Labels which may be partially defined will be denoted by β, with undefined digits indicated by $*$. Thus, $\beta \in \{0, 1, *\}^n$.

At the top of the tree is a node representing a completely undefined label ($\beta = *^n$). In each successive level of the tree, one previously undefined digit of β is set. After descending n levels in the tree, all digits of β will be set. Nodes at this level are leaf nodes, and the fully specified label is the α of basis function χ_α.

3.1.2 Exploring the Search Space

As its name implies, the best-first search algorithm explores regions of the search space in best-first order. Because the goal is to find the labels of the basis functions with the largest coefficients, the "best" regions are those that contain labels of basis functions with large coefficients.

The search algorithm begins at the root node of the search space ($\beta = *^n$). Each internal node of the search is an ancestor of all leaf nodes whose label matches β on each of its defined digits. Therefore, each β represents a region of the Fourier spectrum made up of the labels of those leaf nodes. More precisely, β represents the region of $\hat{f}$ consisting of all $\hat{f}(\alpha)$ such that $\forall i\ (\alpha_i = \beta_i \vee \beta_i = *)$. Thus, the root node of the search represents the entire Fourier spectrum. The notation $\alpha \in \beta$ will be used to denote that α is in region β. In other words, $\alpha \in \beta \longrightarrow \forall i\ (\alpha_i = \beta_i \vee \beta_i = *)$. Thus, β can be thought of as a set of basis function labels.

The largest possible Fourier coefficient of any function is 1. Therefore, when the search begins, the largest possible coefficient is assumed to be 1. The search algorithm selects a digit i and replaces the root node with the two child nodes that result from setting β_i to 0 and 1. Now the search space has been divided in two, with one half representing the region of $\hat{f}$ containing all $\hat{f}(\alpha)$ for which $\alpha_i = 0$, and the other half representing the region of $\hat{f}$ containing all $\hat{f}(\alpha)$ for which $\alpha_i = 1$.

The key to the best-first search algorithm is that for any β it is possible to compute upper bounds on $\max_{\alpha \in \beta} |\hat{f}(\alpha)|$. Thus, regions of $\hat{f}$ can be repeatedly subdivided (by replacing the appropriate node with its child nodes), and bounds on $\max_{\alpha \in \beta} |\hat{f}(\alpha)|$ can be computed for each region. As regions are split into smaller regions, the bounds on $\max_{\alpha \in \beta} |\hat{f}(\alpha)|$ become increasingly tight, until finally, at the leaf nodes, the bound gives the exact value of a single approximated coefficient. The best-first search algorithm finds the largest coefficients by visiting nodes in order of highest upper bound until a solution node is found.

3.1.3 Bounding Regions of the Fourier Spectrum

The magnitude of a Fourier coefficient is proportional to the number of training examples that the corresponding basis function classifies correctly. Thus, in order to bound the largest possible coefficient in a region of the Fourier spectrum, it is necessary to determine that every basis function in that region will misclassify at least a certain fraction of the examples. This section presents a novel method for accomplishing this task.

Conflicting Examples

The coefficient bounding technique is based on finding pairs of examples for which every basis function in a region will classify one and only one of the pair correctly. Such pairs of examples will be called *conflicting examples*.

Definition 1 (Conflicting Examples)**.** *Two examples,* $(x, f(x))$ *and* $(y, f(y))$*, are conflicting examples with respect to basis function* χ_α *if and only if* $(f(x) = f(y)) \wedge (\chi_\alpha(x) \neq \chi_\alpha(y))$ *or* $(f(x) \neq f(y)) \wedge (\chi_\alpha(x) = \chi_\alpha(y))$*. Two examples,* $(x, f(x))$ *and* $(y, f(y))$*, are conflicting examples with respect to a region* β *of the Fourier spectrum if and only if* $(x, f(x))$ *and* $(y, f(y))$ *are conflicting examples with respect to* χ_α *for all* $\alpha \in \beta$*.*

Notice that conflicting examples are always conflicting with respect to a particular basis function or region of the Fourier spectrum. Therefore, two examples can be conflicting with respect to one region, but not to another.

Examples are conflicting with respect to a basis function if that function classifies one and only one of the examples correctly. The "one and only one" aspect of the definition is important because it makes the notion of conflicting examples invariant to basis function inversion. If a basis function misclassifies more than half of the examples then it will be inverted (assigned a negative coefficient) so that it classifies at least half of the examples correctly. If it can be determined, however, that a basis function will classify one and only one of two examples correctly, then it will still classify one and only one of those examples correctly after being inverted.

This means that if conflicting examples can be found, it is certain that the basis function will misclassify examples. Knowing that a specific number of examples are certain to be misclassified, it is possible to bound that basis function's coefficient, without needing to know how it will classify those examples. More importantly, if examples can be found to be conflicting with respect to a region of the Fourier spectrum, it is possible to bound the maximum coefficient of the entire region without needing to know how any of those functions will classify the examples.

Finding Conflicting Examples

A brute-force approach to finding examples that are conflicting with respect to a region of basis functions might test every pair of examples on every basis function in the region. However, since there are potentially an exponential number of basis functions in a region, such an approach would be computationally expensive and would defeat the purpose of the best-first search algorithm. Fortunately, the recursive structure of the Fourier basis allows for a more efficient approach.

This approach is based on the following theorem, which describes conditions under which examples will be conflicting with respect to a particular region of basis functions.

Theorem 1. *Let $\beta \in \{0, 1, *\}^n$ denote a region of the Fourier spectrum, let $(x, f(x))$ and $(y, f(y))$ be any two examples, and let parity(v) be a function that returns 1 if v is even and -1 if v is odd. Then if $\forall i \in \{i|\beta_i = *\}$ $x_i = y_i$ and one of the following is true:*

- *$f(x) = f(y)$ and parity$(\sum_{i \in \{i|\beta_i = 1\}} x_i) \neq$ parity$(\sum_{i \in \{i|\beta_i = 1\}} y_i)$*
- *$f(x) \neq f(y)$ and parity$(\sum_{i \in \{i|\beta_i = 1\}} x_i) =$ parity$(\sum_{i \in \{i|\beta_i = 1\}} y_i)$*

then $(x, f(x))$ and $(y, f(y))$ are conflicting examples with respect to β.

Proof. If $(x, f(x))$ and $(y, f(y))$ are conflicting with respect to β, then for all $\alpha \in \beta$, either $(f(x) = f(y)) \wedge (\chi_\alpha(x) \neq \chi_\alpha(y))$ or $(f(x) \neq f(y)) \wedge (\chi_\alpha(x) = \chi_\alpha(y))$. By

definition, $\chi_\alpha(x) = parity(\sum_{i\in\{i|\alpha_i=1\}} x_i)$ and $\chi_\alpha(y) = parity(\sum_{i\in\{i|\alpha_i=1\}} y_i)$. Therefore, $(x, f(x))$ and $(y, f(y))$ are conflicting with respect to β if for all $\alpha \in \beta$ either $(f(x) = f(y)) \wedge (parity(\sum_{i\in\{i|\alpha_i=1\}} x_i) \neq parity(\sum_{i\in\{i|\alpha_i=1\}} y_i))$ or $(f(x) \neq f(y)) \wedge (parity(\sum_{i\in\{i|\alpha_i=1\}} x_i) = parity(\sum_{i\in\{i|\alpha_i=1\}} y_i))$.

For all $\alpha \in \beta$, $\beta_i = 1 \longrightarrow \alpha_i = 1$ and $\alpha_i = 1 \longrightarrow (\beta_i = 1 \vee \beta_i = *)$. Therefore, $parity(\sum_{i\in\{i|\alpha_i=1\}} x_i) = parity(\sum_{i\in\{i|\beta_i=1\vee(\beta_i=*\wedge\alpha_i=1)\}} x_i)$. However, if $x_i = y_i$ the equivalence of $parity(\sum_{i\in\{i|\beta_i=1\vee(\alpha_i=1\wedge\beta_i=*)\}} x_i)$ and $parity(\sum_{i\in\{i|\beta_i=1\vee(\alpha_i=1\wedge\beta_i=*)\}} y_i)$ is unaffected by the i^{th} input. (If $x_i = y_i$ then either both parities will change or neither parity will change as a result of input i.) Therefore, if $\forall i \in \{i|\beta_i = *\}$ $x_i = y_i$, then those inputs for which $\beta_i = *$ can be ignored, and checking the equivalence of $parity(\sum_{i\in\{i|\beta_i=1\vee(\alpha_i=1\wedge\beta_i=*)\}} x_i)$ and $parity(\sum_{i\in\{i|\beta_i=1\vee(\alpha_i=1\wedge\beta_i=*)\}} y_i)$ is identical to checking the equivalence of $parity(\sum_{i\in\{i|\beta_i=1\}} x_i)$ and $parity(\sum_{i\in\{i|\beta_i=1\}} y_i)$. □

A technique for finding examples that are conflicting with respect to any region β of the Fourier spectrum follows immediately from Theorem 1. For any β, conflicting examples can be identified by checking for pairs of examples that are identical on all undefined digits of β and comparing their difference in parity over the those inputs for which $\beta_i = 1$ to their difference in output. If the differences in parity and output do not match appropriately then the examples are conflicting.

Bounding Coefficients via Conflicting Examples

Once pairs of examples that are conflicting with respect to a region of the Fourier spectrum have been found, it is possible to place an upper bound on the largest approximate coefficient in that region. Theorem 2 shows how the largest possible coefficient in a region of the Fourier spectrum is bounded by the number of pairs of examples that are conflicting with respect to that region. Its proof is based on the following two lemmas.

Lemma 1. *Let $\tilde{\hat{f}}(\alpha)$ be the approximated coefficient of Fourier basis function χ_α, and let X be the set of training examples from which $\tilde{\hat{f}}(\alpha)$ is approximated. Then*

$$\tilde{\hat{f}}(\alpha) = 1 - \frac{2d}{|X|} \tag{3.1}$$

where d is the number of examples in X for which $\chi_\alpha(x) \neq f(x)$.

Proof. The formula given in Chapter 2 (Equation 2.11) for approximating Fourier coefficients from a training set is the following:

$$\tilde{\hat{f}}(\alpha) = \frac{1}{|X|} \sum_{x \in X} f(x)\chi_\alpha(x)$$

Notice that for every $x \in X$ such that $f(x) = \chi_\alpha(x)$, the sum in the above formula increases by 1 because $(1 \cdot 1) = (-1 \cdot -1) = 1$. For every $x \in X$ such that $f(x) \neq \chi_\alpha(x)$, the sum decreases by 1 because $(1 \cdot -1) = (-1 \cdot 1) = -1$. Therefore, the sum is equal to the number of examples for which $f(x) = \chi_\alpha(x)$ minus the number of examples for which $f(x) \neq \chi_\alpha(x)$. Let c be the number of examples for which $f(x) = \chi_\alpha(x)$, and let d be the number of examples for which $f(x) \neq \chi_\alpha(x)$. Then the formula for approximating coefficients can be expressed in terms of c and d as follows:

$$\tilde{\hat{f}}(\alpha) = \frac{1}{|X|}(c - d)$$

Observe that $c + d = |X|$, so that $c = |X| - d$. By replacing c with $|X| - d$ and rearranging the right-hand side of the equation, the desired result is obtained:

$$\begin{aligned}
\tilde{\hat{f}}(\alpha) &= \frac{1}{|X|}(c - d) \\
&= \frac{1}{|X|}((|X| - d) - d) \\
&= \frac{1}{|X|}(|X| - 2d) \\
&= 1 - \frac{2d}{|X|}
\end{aligned}$$

□

Lemma 1 shows how the number of examples that a basis function misclassifies affects its signed coefficient value. However, for learning purposes, large negative correlations are just as useful as large positive correlations. Consequently, the absolute value of a coefficient is more important than its signed value. Lemma 2 expresses the idea of Lemma 1 in terms of the absolute value of a coefficient.

Lemma 2. *Let $\tilde{\hat{f}}(\alpha)$ be the approximated coefficient of Fourier basis function χ_α, and let X be the set of training examples from which $\tilde{\hat{f}}(\alpha)$ is approximated. Then*

$$|\tilde{\hat{f}}(\alpha)| = 1 - \frac{2\min(c,d)}{|X|} \tag{3.2}$$

where c is the number of examples in X for which $\chi_\alpha(x) = f(x)$ and d is the number of examples in X for which $\chi_\alpha(x) \neq f(x)$.

Proof. If $\tilde{\hat{f}}(\alpha) \geq 0$, then by Lemma 1, $1 - \frac{2d}{|X|} \geq 0$. If $1 - \frac{2d}{|X|} \geq 0$, then $d \leq \frac{|X|}{2}$. Since $|X| = c + d$, if $d \leq \frac{|X|}{2}$ then $c \geq \frac{|X|}{2}$ and $\min(c,d) = d$. Thus, when $\tilde{\hat{f}}(\alpha) \geq 0$, $1 - \frac{2\min(c,d)}{|X|} = 1 - \frac{2d}{|X|} = \tilde{\hat{f}}(\alpha)$.

If $\tilde{\hat{f}}(\alpha) < 0$, then by Lemma 1, $1 - \frac{2d}{|X|} < 0$. If $1 - \frac{2d}{|X|} < 0$, then $d > \frac{|X|}{2}$, and $\min(c,d) = c$. Therefore, when $\tilde{\hat{f}}(\alpha) < 0$, $1 - \frac{2\min(c,d)}{|X|} = 1 - \frac{2c}{|X|}$. The following steps show that $1 - \frac{2c}{|X|} = -(1 - \frac{2d}{|X|}) = -\tilde{\hat{f}}(\alpha)$, completing the proof:

$$\begin{aligned}
1 - \frac{2c}{|X|} &= 1 - \frac{2(|X| - d)}{|X|} \\
&= 1 - \frac{2|X|}{|X|} + \frac{2d}{|X|} \\
&= 1 - 2 + \frac{2d}{|X|} \\
&= -1 + \frac{2d}{|X|} \\
&= -\left(1 - \frac{2d}{|X|}\right) = -\tilde{\hat{f}}(\alpha)
\end{aligned}$$

□

By combining Lemma 2 with the notion of conflicting examples, it is possible to compute bounds on the largest Fourier coefficient in any region of the Fourier spectrum. The key is that for every pair of conflicting examples, c and d both increase by one for every basis function in the region, making it possible to bound the coefficients without needing to know whether c or d will be smaller for any particular basis function in the region.

Theorem 2. *Let $\beta \in \{0, 1, *\}^n$ denote a region of approximated Fourier spectrum $\tilde{\hat{f}}$, and let X be the set of training examples from which $\tilde{\hat{f}}$ is approximated. Then*

$$\max_{\alpha \in \beta} |\tilde{\hat{f}}(\alpha)| \leq 1 - \frac{2p}{|X|} \tag{3.3}$$

where p is the number of pairs of examples $((x, f(x)), (y, f(y)))$ in X that are conflicting with respect to β. (No example can be included in more than one pair.)

Proof. By Lemma 2, for any χ_α in region β, $|\tilde{\hat{f}}(\alpha)|$ is given by the following:

$$|\tilde{\hat{f}}(\alpha)| = 1 - \frac{2\min(c, d))}{|X|}$$

where c is the number of examples for which $\chi_\alpha(x) = f(x)$ and d is the number of examples for which $\chi_\alpha(x) \neq f(x)$.

The number of pairs of examples that are conflicting with respect to β is given by p. By definition, if a pair of examples are conflicting with respect to a region β, then every χ_α in region β will classify one of the examples correctly and misclassify the other. Thus, for any χ_α in region β, $c \geq p$ and $d \geq p$. Therefore, $\min(c, d) \geq p$, and the following holds:

$$|\tilde{\hat{f}}(\alpha)| \leq 1 - \frac{2\min(c, d))}{|X|} \leq 1 - \frac{2p}{|X|}$$

□

Based on Theorems 1 and 2, it is possible to implement an efficient best-first search algorithm for finding large coefficients. Before describing this algorithm, however, the following section shows how the largest possible coefficient in a region can be represented implicitly by a set of examples associated with the region. This technique allows the algorithm to avoid redundant calculations by passing previously gained information from parent nodes to their children.

3.1.4 Avoiding Redundant Calculations

Because of the recursive structure of the Fourier basis, it is possible to improve the efficiency of spectral computations by avoiding redundant calculations. If a pair of examples are conflicting with respect to region β of the Fourier spectrum, then they are conflicting with respect to every subregion of β. Therefore, as long as the reduction in coefficient bound due to previously discovered conflicting examples is remembered, those examples do not need to be reconsidered when computing bounds on subregions.

One way to take advantage of this fact is to store with each node of the search the set of examples that are not conflicting with the region that the node represents. This way, when a parent node is replaced by its children, only the examples in the parent need to be examined to find pairs of examples that are conflicting with respect to each new subregion.

The root node of the search, which represents the entire Fourier spectrum, contains every training example. When it, and any future node, is replaced by its children, each child initially receives all of its parent's examples. Then, any pairs of examples that are conflicting with respect to the child's subregion are removed.

Interestingly, by storing all non-conflicting examples with each node of the search space, the largest possible coefficient of any basis function within the region that a node represents is bounded by the number of examples at that node. The following theorem describes the relationship.

Theorem 3. *Let $\beta \in \{0, 1, *\}^n$ denote a region of the Fourier spectrum, let X be the set of all training examples, and let $X_\beta \subseteq X$ be a set of examples derived from X as follows: X_β begins with all of the examples in X, after which examples that are conflicting with respect to β are removed one pair at a time until no two examples remaining in X_β are conflicting with respect to β. Then*

$$\max_{\alpha \in \beta} |\tilde{\hat{f}}(\alpha)| \leq \frac{|X_\beta|}{|X|} \tag{3.4}$$

Proof. By Theorem 2,

$$\max_{\alpha \in \beta} |\tilde{\hat{f}}(\alpha)| \leq 1 - \frac{2p}{|X|}$$

where p is the number of pairs of examples in X that are conflicting with respect to β (with no example included in more than one pair). This inequality can be rewritten as follows:

$$\begin{aligned} \max_{\alpha \in \beta} |\tilde{\hat{f}}(\alpha)| &\leq 1 - \frac{2p}{|X|} \\ &\leq \frac{|X|}{|X|} - \frac{2p}{|X|} \\ &\leq \frac{|X| - 2p}{|X|} \end{aligned}$$

Since X_β is obtained from X by iteratively removing pairs of conflicting examples until no two examples in X_β are conflicting, and since each pair of examples removed reduces $|X_\beta|$ by 2, $|X_\beta| = |X| - 2p$. Replacing $|X| - 2p$ with $|X_\beta|$ in the previous inequality gives the desired result. □

Thus, the bound on the largest coefficient in every region of the Fourier spectrum can be represented implicitly by storing non-conflicting examples with each node. Although storing this set of examples improves computational efficiency, it also increases memory complexity. Every node on the frontier of the search is stored in memory, and each now has its own set of examples. However, despite the fact that this is costly, as the search progresses to deeper levels of the search tree there are increasingly many pairs of conflicting examples. Consequently, the number of examples stored at each node decreases as the search descends deeper into the search tree.

The technique of storing non-conflicting examples with each node is not required, and it does not need to be used if the data set and/or search frontier is too large or if reduced memory usage is more important than computation time. Instead of implicitly representing the coefficient bound by the number of training examples, a single value could be stored with each node to indicate this bound. The computational complexity would increase, however, as each time a node was split into child nodes the entire original data set would need to be examined again to find conflicting examples.

In addition to providing a bound on the largest coefficient in each region of the search space, by storing examples with the nodes of the search, the number of examples at leaf nodes is directly proportional to the absolute value of the actual coefficient that the leaf node represents. The following theorem expresses this idea.

Theorem 4. *Let $\alpha \in \{0,1\}^n$ denote the label of Fourier coefficient $\tilde{\hat{f}}(\alpha)$, let X be the set of all training examples, and let $X_\alpha \subseteq X$ be a set of examples derived from X as follows: X_α begins with all of the examples in X, after which examples that are conflicting with respect to α are removed one pair at a time until no two examples*

remaining in X_α are conflicting with respect to α. Then

$$|\tilde{\hat{f}}(\alpha)| = \frac{|X_\alpha|}{|X|}$$

Proof. By Theorem 3, $|\tilde{\hat{f}}(\alpha)| \leq \frac{|X_\alpha|}{|X|}$. To show that $|\tilde{\hat{f}}(\alpha)| = \frac{|X_\alpha|}{|X|}$, it must be shown that χ_α will classify all examples in X_α correctly.

By definition, X_α cannot contain any pairs of examples that are conflicting with respect to χ_α. Thus, for every pair of examples in X_α, χ_α must be able to classify both examples correctly, although it may need to be inverted to do so. Since χ_α must either be inverted or not inverted for all pairs of examples, it must be shown that for every pair of examples in X_α, χ_α will either classify all examples correctly when inverted or classify all examples correctly without being inverted.

According to Theorem 1, two examples are conflicting if $\forall i \in \{i|\alpha_i = *\}$ $x_i = y_i$ and one of the following is true:

- $f(x) = f(y)$ and $parity(\sum_{i\in\{i|\alpha_i=1\}} x_i) \neq parity(\sum_{i\in\{i|\alpha_i=1\}} y_i)$
- $f(x) \neq f(y)$ and $parity(\sum_{i\in\{i|\alpha_i=1\}} x_i) = parity(\sum_{i\in\{i|\alpha_i=1\}} y_i)$

Since, α is a completely defined label, however, α_i is never equal to $*$, and the first condition ($\forall i \in \{i|\alpha_i = *\}$ $x_i = y_i$) will be satisfied for every pair of examples. Thus, for every pair of examples $((x, f(x)), (y, f(y))) \in X_\alpha$, $(x, f(x))$ and $(y, f(y))$ are conflicting if one of the other conditions is true. Since X_α contains no conflicting pairs of examples, it must be true that for any $((x, f(x)), (y, f(y))) \in X_\alpha$ one of the following is true:

- $f(x) = f(y)$ and $parity(\sum_{i\in\{i|\alpha_i=1\}} x_i) = parity(\sum_{i\in\{i|\alpha_i=1\}} y_i)$
- $f(x) \neq f(y)$ and $parity(\sum_{i\in\{i|\alpha_i=1\}} x_i) \neq parity(\sum_{i\in\{i|\alpha_i=1\}} y_i)$

This means that all positive examples will have the same parity over the inputs for which $\alpha_i = 1$, and all negative examples will have the opposite parity over those inputs. Since all examples of the same class have the same parity over the inputs for which $\alpha_i = 1$, while those of the other class have the opposite parity, basis function

χ_α, which computes the parity of those inputs, will be able to classify all examples in X_α correctly. □

Thus, it is possible to determine the absolute value of any coefficient $\tilde{\hat{f}}(\alpha)$ simply by removing pairs of examples that are conflicting with respect to χ_α. The best-first search algorithm for finding the largest coefficients uses this technique.

3.1.5 A Best-First Search for Large Coefficients

The best-first search algorithm for finding the largest Fourier coefficients is illustrated in Figures 3.2 and 3.3. The algorithm is divided into two procedures: `FindLargestCoefficients` and `CreateChild`.

The `FindLargestCoefficients` Procedure

The algorithm begins with a call to `FindLargestCoefficients`. This procedure takes two inputs: X, a set of training examples, and r, a number of basis function labels to find. The procedure returns R, the set of basis function labels corresponding to the r largest coefficients. (If there is a tie for the r^{th} largest coefficient, any of them may be returned.)

As described in Sections 3.1.1 and 3.1.2, the search can be thought of as an exploration through a binary tree of nodes representing partially and completely defined basis function labels (see Figure 3.1). The leaf nodes of the search tree represent the 2^n completely defined basis function labels.

Every node of the search algorithm contains a basis function label β and a set of training examples X_β. The algorithm begins with the root node of the tree, which contains a completely undefined label and all training examples. This node is initialized at lines 1 and 2. At line 3, this node is inserted into a priority queue, which always places the node with the most examples at the front of the queue.[1] As

[1] If two nodes have the same number of examples, then the node whose label has fewer undefined digits is placed first in the queue, favoring nodes that are closer to a solution. If two nodes have the same number of examples and the same number of undefined digits, then the node whose label contains fewer 1s is placed first, favoring low-order basis functions over high-order basis functions. If there is still a tie, the choice of order is arbitrary.

Figure 3.2: The `FindLargestCoefficients` procedure of the best-first search algorithm. It takes as input a set of examples X and a number of basis functions to find r, and returns R, the set of labels of the r basis functions with the largest coefficients.

```
      FindLargestCoefficients(X, r)
(1)     rootNode.β ← *^n
(2)     rootNode.X_β ← X
(3)     priorityQueue.Insert(rootNode)
(4)     R ← {}
(5)     while |R| < r
(6)       currentNode ← priorityQueue.RemoveFront()
(7)       if ∀i (currentNode.β_i ≠ *) then R ← R ∪ currentNode.β
(8)       else
(9)         i ← SelectDigit(currentNode)
(10)        priorityQueue.Insert(CreateChild(currentNode, i, 0))
(11)        priorityQueue.Insert(CreateChild(currentNode, i, 1))
(12)    return R
```

described in the previous section, the number of examples at a node is proportional to the largest possible coefficient in the region that the node represents. Therefore, the priority queue stores nodes in order of largest possible coefficient. This ensures that the search space is explored in "best-first" order.

At line 4, the set R, which will be returned at line 12 once it contains the labels of the r largest coefficients, is initialized as an empty set. The major portion of the algorithm is contained in the while loop of lines 5-11 which implements the best-first search. It will repeat until the labels of the r largest coefficients have been found. In each iteration of the loop, the node at the front of the queue is removed and is either added to the solution (line 7) or replaced by its child nodes (lines 9-11). Since the node at the front of the queue always has the highest coefficient bound, the node at the front of the queue represents either the largest coefficient or the region that could potentially contain the largest coefficient.

A node is a solution node if its label is completely defined, meaning that it now represents a specific basis function label. If a node's label is not completely defined then it represents a region of the Fourier spectrum, and its children, which represent two halves of the region, will be inserted into the queue. These child nodes are created

Figure 3.3: The `CreateChild` procedure of the best-first search algorithm. It takes as input a node of the best-first search, an index i, and a value s, and returns the child node that results from setting β_i to s.

```
      CreateChild(parent, i, s)
(13)    child.X_β ← parent.X_β
(14)    child.β ← parent.β
(15)    child.β_i ← s
(16)    if s = 1 then
(17)      for each (x, f(x)) ∈ child.X_β
(18)        if x_i = 1 then (x, f(x)) ← (x, −f(x))
(19)    for each (x, f(x)) ∈ child.X_β
(20)      if ∃(y, f(y)) ∈ child.X_β such that ∀j ∈ {j|β_j = ∗} x_j = y_j then
(21)        if f(x) ≠ f(y) then child.X_β ← child.X_β − {(x, f(x)), (y, f(y))}
(22)    return child
```

by calls to the `CreateChild` procedure (lines 10-11). As described in Section 3.1.1, each child node's label is identical to its parent except that one additional digit is set. The same digit is set in both children, with one child's label receiving a value of 0 for that digit while the other receives a value of 1.

The digit that will be set in the child nodes' labels is determined by the `SelectDigit` procedure (line 9). Any unspecified digit of the label can be selected. A naive approach might set digits in a predefined order. However, an intelligent choice of digit selection can drastically reduce the number of nodes that must be visited to find the largest coefficients. A heuristic for digit selection that has proven very effective is to select the digit that minimizes the following:

$$\arg\min_i \left(\min \left(\max_{\alpha \in \beta_{i \leftarrow 0}} |\tilde{\hat{f}}(\alpha)|, \max_{\alpha \in \beta_{i \leftarrow 1}} |\tilde{\hat{f}}(\alpha)| \right) \right)$$

where $\beta_{i \leftarrow 0}$ and $\beta_{i \leftarrow 1}$ denote the regions that result from setting the i^{th} digit of the current region to 0 and 1, respectively. This formula returns the digit that will cause the greatest reduction in coefficient bound in either child node.

The intuition behind this heuristic is that larger decreases in coefficient bound indicate more information gain. Consider the worst case, in which the bound does not decrease for either child. In this case, the algorithm has been given no useful

information regarding the maximum coefficient down each branch of the search tree.

Since each child's label is more specific than its parent's, pairs of examples may be conflicting with respect to each child's region that were not conflicting with the parent's. The `CreateChild` procedure is responsible for removing such pairs of examples so that when the child nodes are inserted into the queue their coefficient bounds will have been appropriately lowered.

The `CreateChild` Procedure

The `CreateChild` procedure takes as input an internal node of the search algorithm, the index i of an unspecified digit of the node's label, and a value $s \in \{0, 1\}$ to assign to the digit. The procedure returns the child node that results from setting the i^{th} digit of its label to s. Besides having an additional digit of its label set, the child node returned from the `CreateChild` procedure will also contain all of its parent's examples that are not conflicting with respect to its more specific region.

The `CreateChild` procedure begins by initializing the child node with all of its parent's training examples (line 13), as well as its parent's label (line 14). Then, at line 15, digit i of the child's label is set to s. The remainder of the procedure is concerned with finding and removing pairs of examples that are conflicting with respect to the child node's region.

In lines 19-21, the `CreateChild` procedure searches through the examples to find pairs of conflicting examples. For each example $(x, f(x))$, the algorithm checks for a potentially conflicting example. If there exists an example $(y, f(y))$ such that $x_j = y_j$ for all unspecified β_j, then the examples could be conflicting. Notice that this is one of the conditions for conflicting examples given in Theorem 1. The other requirement for two examples to be conflicting is that either their outputs are different but their parities over the specified digits of β are the same or their outputs are the same but their parities are different. This requirement is checked on line 21. Because of modifications made to the examples in lines 16-18, this requirement can be checked simply by checking whether $f(x) \neq f(y)$.

The check for the equality of $f(x)$ and $f(y)$ at line 21 is sufficient to determine

whether two examples are conflicting because lines 16-18 of the `CreateChild` procedure cause the parity information of the examples to be stored implicitly with each example. If `CreateChild` is called with $s = 1$, then at lines 17-18 the outputs of all examples for which $x_i = 1$ are inverted. The proof of the following theorem reveals why this method allows conflicting examples to be correctly identified.

Theorem 5. *Let $\beta \in \{0, 1, *\}^n$ denote a region of the Fourier spectrum, let X be a set of training examples, and let $X_\beta \subseteq X$ be a set of examples derived from X as follows:*

$$
\begin{aligned}
&X_\beta \leftarrow X \\
&\textit{for each } i \in \{i|\beta_i = 1\} \\
&\quad \textit{for each } (x, f(x)) \in X_\beta \\
&\quad\quad \textit{if } x_i = 1 \textit{ then } (x, f(x)) \leftarrow (x, -f(x))
\end{aligned}
$$

Then for any $((x, f(x)), (y, f(y)) \in X_\beta$, if $\forall i \in \{i|\beta_i = *\}$ $x_i = y_i$ and $f(x) \neq f(y)$ then $(x, f(x))$ and $(y, f(y))$ are conflicting with respect to β.

Proof. Let $(x, f(x))$ and $(y, f(y))$ be any two examples such that $\forall i \in \{i|\beta_i = *\}$ $x_i = y_i$. According to Theorem 1, $(x, f(x))$ and $(y, f(y))$ are conflicting with respect to β if $f(x) = f(y)$ and $parity(\sum_{i\in\{i|\beta_i=1\}} x_i) \neq parity(\sum_{i\in\{i|\beta_i=1\}} y_i)$ or if $f(x) \neq f(y)$ and $parity(\sum_{i\in\{i|\beta_i=1\}} x_i) = parity(\sum_{i\in\{i|\beta_i=1\}} y_i)$. Since the outputs of $(x, f(x))$ and $(y, f(y))$ are inverted once for each input $i \in \{i|\beta_i = 1\}$ such that $x_i = 1$ and $y_i = 1$, respectively, the parities of $\sum_{i\in\{i|\beta_i=1\}} x_i$ and $\sum_{i\in\{i|\beta_i=1\}} y_i$ determine whether the outputs of $(x, f(x))$ and $(y, f(y))$are inverted in X_β. If the parity is even, then an example's output is the same in X_β; if it is odd, then the output is inverted. Consequently, if $f(x) = f(y)$ and $parity(\sum_{i\in\{i|\beta_i=1\}} x_i) \neq parity(\sum_{i\in\{i|\beta_i=1\}} y_i)$, then the examples will have opposite outputs in X_β. Furthermore, if $f(x) \neq f(y)$ and $parity(\sum_{i\in\{i|\beta_i=1\}} x_i) = parity(\sum_{i\in\{i|\beta_i=1\}} y_i)$ then the examples will have opposite outputs in X_β. In either case, if $(x, f(x))$ and $(y, f(y))$ are conflicting then they will have opposite outputs in X_β.

□

If examples' outputs were not inverted as described above, then each time two examples were checked for a potential conflict it would be necessary to look back at previously examined inputs to determine their parities over those inputs. However, since the outputs of the examples implicitly store the necessary parity information it suffices to examine only their outputs. Each time `CreateChild` is called, the i^{th} input of each example is examined (if $s = 1$), but by applying this inversion technique, that input will not need to be looked at again by any children of the current node.

This improvement in run-time efficiency is another benefit of storing examples with each node. As before, however, this optimization is optional. If memory limitations make storing examples at the nodes infeasible, then instead of inverting outputs to implicitly store parity information the parities can be recomputed each time `CreateChild` is called.

Analysis

The Fourier-based techniques presented in the learning theory community avoid exponential complexity by employing probabilistic algorithms that always terminate in polynomial time but only find the largest coefficients with high probability. Furthermore, the guarantees of correctness depend on the target function belonging to a specific class of functions. In contrast, as expressed in the following theorem, the best-first search algorithm presented here will always find the largest coefficients[2] of any function.

Theorem 6. *Given a set of examples X of an n-input Boolean function f, and a number r, $1 \leq r \leq 2^n$, the* ***`FindLargestCoefficients`*** *procedure returns a set $R \subseteq \{0,1\}^n$ of basis function labels having the following properties:*

1. $|R| = r$
2. $\alpha \in R \longrightarrow \forall \beta \notin R \; (|\tilde{\hat{f}}(\alpha)| \geq |\tilde{\hat{f}}(\beta)|)$

[2]More precisely, the algorithm always finds the largest approximated coefficients, as they are defined in the previous chapter. The approximated coefficients are the best estimates of the true coefficients based on available training data.

Proof. For every execution of the procedure, R is initially empty (line 4) when the while-loop of lines 5-11 is entered. Since at most one label is added to R each time through the loop (line 7), execution will leave the while-loop after the r^{th} label is added, and R can never contain more than r labels.

Labels are added to R at line 7 when the label of the node at the front of the queue is fully-specified. As nodes in the priority queue are replaced by their children, all 2^n fully-specified labels will eventually appear in a node at the front of the queue. Therefore, for any r, $1 \leq r \leq 2^n$, there will eventually be r labels added to R.

It remains to show that the second condition, that the coefficient of every basis function whose label is in R is at least as large as the coefficient of any basis function whose label is not in R, will always be satisfied.

In the `FindLargestCoefficients` procedure, nodes are removed from the priority queue in order of $|X_\beta|$, the number of examples at a node. The initial node of the search contains all training examples. According to Theorem 5, the `CreateChild` procedure ensures that in each subsequent node any pairs of examples that are conflicting with respect to that node's label are removed. Therefore, in each node, X_β contains all examples that are not conflicting with respect to that node's label β. Theorem 3 shows that the quantity $|X_\beta|/|X|$ bounds the largest coefficient in region β, while according to Theorem 4, if $\beta \in \{0,1\}^n$ then the quantity $|X_\beta|/|X|$ is the exact coefficient of basis function χ_α, where $\alpha = \beta$. Since the quantity $|X|$ is a positive constant, storing nodes in order of $|X_\beta|$ is equivalent to storing nodes in order of $|X_\beta|/|X|$. Therefore, the priority queue stores nodes in order of largest possible coefficient.

Since nodes are removed from the priority queue in order of largest possible coefficient, the requirement that $\alpha \in R \longrightarrow \forall \beta \notin R\ (|\tilde{\hat{f}}(\alpha)| \geq |\tilde{\hat{f}}(\beta)|)$ is enforced by the priority queue. When a node with a fully-specified label arrives at the front of the queue, the coefficient of the basis function it represents must be at least as large as the coefficient of any basis function represented by nodes remaining in the queue. □

In exchange for the guarantee of always finding the largest coefficients, there is no guarantee that the best-first search algorithm will run in polynomial time. In fact,

in the worst case, the algorithm would require exponential time to find the largest coefficients.

There are $2^{n+1} - 1$ nodes in the search tree, of which 2^n are leaf nodes and $2^n - 1$ are internal nodes. For each leaf node encountered there is a constant-time operation, as the node's label is added to the solution. For each internal node encountered the `CreateChild` procedure will be called twice, although the second execution does not affect the time complexity. The time complexity of the `CreateChild` procedure is dominated by the loop of lines 19-21. If a node contains k examples, and examples are stored in a structure that allows $\log k$ lookups, then this loop will require $O(k \log k)$ time (for each example, a $\log k$ lookup is performed to search for a potential conflicting example). Since the `CreateChild` procedure could be called 2^n times (ignoring constant terms), the time complexity of the entire algorithm is bounded by $O(2^n k \log k)$.

Notice that the $O(2^n k \log k)$ bound describes worst-case performance. However, the number of nodes in the search space that must be expanded before a solution is found is problem dependent, and can vary between n and $2^n - 1$. (A node expansion refers to a node being replaced by its child nodes.) Therefore, it can be more meaningful to describe the time complexity as $O(mk \log k)$, where m is the number of nodes expanded during the search. Since the number of node expansions is problem dependent, a better idea of expected real-world performance can be obtained by testing the algorithm on real-world problems. An empirical analysis of real-world performance is presented in the following section.

Memory usage is also dependent on the size of the search. Every node on the frontier of the search is stored in memory. Initially there is one node on the frontier. Thereafter, each node expansion results in one node being replaced by two, so the number of nodes on the frontier of the search increases by one for each node expanded. When leaf nodes are removed from the frontier, however, no additional nodes are added, so the number of nodes on the frontier may be less than the number of nodes expanded. If training examples are stored at the nodes, then the memory complexity is $O(mk)$, where m is the number of nodes expanded and k is the number

of training examples. No more than 2^{n-1} nodes could be on the search frontier at a time, so the worst-case memory complexity is $O(2^n k)$.

As mentioned previously, as the search descends to deeper levels of the tree, the number of examples per node becomes increasingly small. This does not affect the big-O complexity, but it is helpful in practice for both memory usage and run time. It was also mentioned previously that examples do not need to stored with the nodes. If they are not, then memory usage per node becomes constant (the nodes only need to store a label and a coefficient bound), so the memory complexity drops to $O(m)$, with a worst-case bound of $O(2^n)$. By not storing examples at the nodes the worst-case time complexity will increase to $O(m(nk+k\log k))$, which in the worst case is $O(2^n(nk + k\log k))$. The increase in complexity is due to the fact that parity information that would be stored implicitly in each node must now be recomputed each time `CreateChild` is called.

3.2 Real-World Results

Although the worst-case time complexity of the best-first search algorithm is exponential in the number of inputs, the worst-case bound would only provide a meaningful description of real-world performance if real-world problems tended to exemplify worst-case scenarios for the search algorithm. This section attempts to give a better idea of the expected real-world performance of the algorithm by presenting results of experiments with several real-world problems.

3.2.1 Node Expansion Analysis

The number of nodes expanded while searching for the largest coefficients can vary between n and $2^n - 1$. Fortunately, for every real-world data set tested, only a small fraction of the nodes needed to be expanded to find the most highly correlated functions.

Table 3.1 shows the number of node expansions that are required to find the basis functions that are most highly correlated with several real-world data sets,

compared to the total number of nodes in the search space. The data sets represent Boolean classification problems and can be found in the UCI machine learning repository [21]. In several cases, non-Boolean input features were encoded as Boolean features. (A more detailed description of the data sets can be found in the appendix.)

Table 3.1: Nodes expanded to find the 1, 10, 100, and 1,000 most highly correlated Fourier basis functions, compared to the total number of nodes in the search space. The number of features for each data set is shown in parentheses. (There is no data for finding 1,000 basis functions for the Pima data set because it is an 8-input problem for which there are only 256 basis functions.)

Data Set	1	10	100	1,000	Total Possible
Chess (37)	209	254	531	2,010	137,438,953,471
German (24)	197	312	820	3,378	16,777,215
Heart (16)	73	111	616	2,804	65,535
Pima (8)	8	23	151	n/a	255
SPECT (22)	1,115	29,521	66,414	114,868	4,194,303
Voting (16)	16	63	339	2,514	65,535
Wisc 1 (36)	540	5,858	38,315	126,474	68,719,476,735
Wisc 2 (33)	322	398	1,060	3,621	8,589,934,591
Wisc 3 (30)	44	80	340	2,445	1,073,741,823

The results in Table 3.1 are interesting. For each of the data sets, only a small portion of the search space needs to be explored to find the most highly correlated functions. When finding the single most highly correlated function, two of the data sets (Pima and Voting) exhibited best-case performance, as only n nodes were expanded. Having only n nodes expanded indicates that the search algorithm was able to go straight from the root node to a solution and conclude that it had found the largest coefficient without ever needing to backtrack.

Even when finding many highly correlated functions, the number of node expansions on each data set is small relative to the total number of possible node expansions. Even for the relatively more difficult SPECT problem, for example, the 114,868 nodes expanded after finding 1,000 basis functions represents only 2.7% of the total search space. These results suggest that typical real-world problems may

more closely represent best-case scenarios for the search algorithm than worst-case scenarios.

3.2.2 Run Time Analysis

The benefit of the algorithm is also apparent when comparing its run time to the time required to compute the Fourier spectrum with the Fast Walsh Transform (FWT). The Fast Walsh Transform is a Boolean analogue to the more general Fast Fourier Transform of discrete-valued functions. The FWT computes the entire Fourier spectrum in $O(n2^n)$ time.

Table 3.2 shows the time required to find the most highly correlated basis functions, compared to the time required to compute the FWT. (Note that for the Chess, Wisc 1, and Wisc 2 datasets the standard FWT algorithm requires more memory than can be allocated and addressed on a 32-bit workstation. The run times shown are estimates based on observing the growth in run time as n increases.)

Table 3.2: Time required to find the 1, 10, 100, and 1,000 most highly correlated basis functions, compared to the time required to compute the Fast Walsh Transform (FWT). The number of attributes for each data set is shown in parentheses. (There is no data for finding 1,000 basis functions for the Pima data set because it is an 8-input problem for which there are only 256 basis functions.)

Data set	1	10	100	1,000	FWT
Chess (37)	.5 s	.6 s	.8 s	1.2 s	11+ hrs
German (24)	.2 s	.2 s	.4 s	.8 s	4.5 s
Heart (16)	< .1 s	< .1 s	< .1 s	< .1 s	< .1 s
Pima (8)	< .1 s	< .1 s	< .1 s	n/a	< .1 s
SPECT (22)	.3 s	3.2 s	5.1 s	6.4 s	1.0 s
Voting (16)	< .1 s	< .1 s	< .1 s	< .1 s	< .1 s
Wisc 1 (36)	.3 s	2.9 s	14.8 s	37.6 s	5+ hrs
Wisc 2 (33)	< .1 s	< .1 s	.1 s	.2 s	45 min
Wisc 3 (30)	< .1 s	< .1 s	< .1 s	.2 s	5 min

For small values of n, there is little benefit to using the best-first search algorithm since the entire spectrum can be computed very quickly by the FWT. For the

larger data sets, however, the exponential complexity of the FWT begins to manifest itself in long run times, while the best-first search algorithm can still find thousands of the most correlated basis functions in seconds.

3.2.3 n vs. m

The exponential worst-case time complexity of the search algorithm and the exponential size of the search space would seem to suggest that as the number of inputs n increases the number of node expansions m would increase exponentially. However, the experiments with these real-world problems suggests a more favorable relationship.

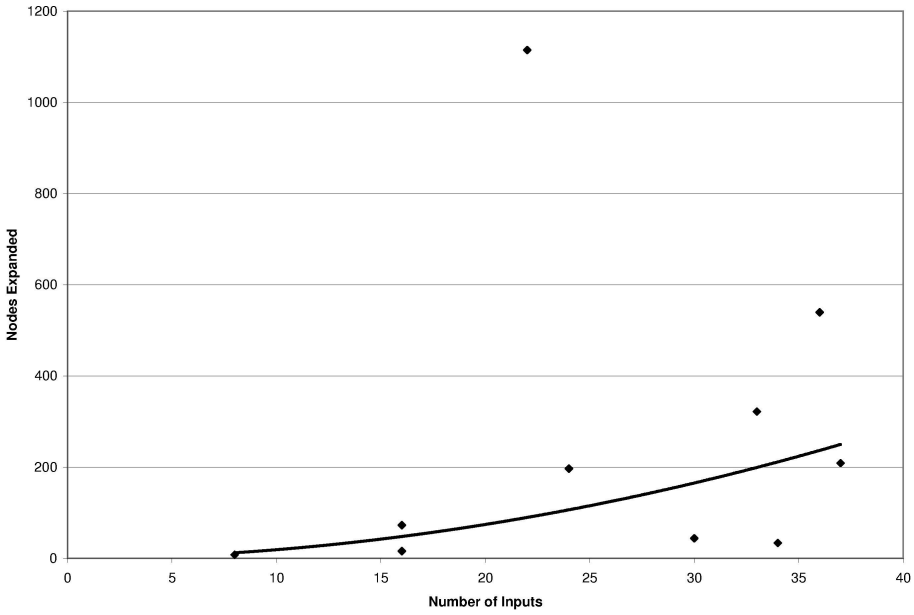
A least-squares fit of a trendline to a plot of m vs. n for these data sets reveals that a single-term polynomial function fits the data better than an exponential function. The R^2 value of the best-fit single-term polynomial is 0.37, while the R^2 value of the best-fit exponential is 0.29. The single-term polynomial is also a better fit than the best-fit linear, multi-term polynomial, and logarithmic equations. Figure 3.4 shows a plot of m vs. n for these data sets when finding the single most highly correlated basis function. The trendline on the graph is the single-term polynomial function that best fit the data points.

The single-term polynomial of the trendline in Figure 3.4 has an exponent of approximately 1.97, indicating that m is approximately proportional to n^2. As the number of basis functions found increases, however, the exponent of the polynomial decreases. A similar trendline for the case in which 1,000 basis functions are found has an exponent of 0.3, suggesting a sub-linear relationship between n and m. These observed relationships suggest that the algorithm will scale well to larger learning problems.

3.3 Recursively Computing the Fourier Spectrum

The techniques used by the best-first search algorithm presented in this chapter can also be used in a recursive variant that can efficiently compute the entire Fourier spectrum.

Figure 3.4: A plot of the number of nodes expanded to find the most highly correlated basis function vs. the number of inputs. The trendline is a least squares fit of a single-term polynomial to the data.



The recursive algorithm is very similar to the best-first search algorithm. The primary difference is that it performs a depth-first traversal of the search space instead of a best-first traversal. This recursive method is illustrated in Figure 3.5, which gives an algorithm for recursively computing the entire Fourier spectrum. The algorithm begins with a call to the `ComputeFourierSpectrum` procedure. The inputs to the procedure are a set X of training examples, a basis function label α, and the index i of the first undefined digit of α. The initial call to `ComputeFourierSpectrum` is made with the original training set X_0, a completely undefined label ($\alpha = *^n$), and index $i = 0$.

The `ComputeFourierSpectrum` procedure makes two calls to the procedure `ComputeFourierSpectrum2` (lines 1-2). The arguments to this procedure are identical to the arguments to `ComputeFourierSpectrum`, with one addition: a value s to assign to α_i. The two calls to `ComputeFourierSpectrum2` at lines 1 and 2 account for the two possible values to assign to α_i.

Figure 3.5: A recursive method for computing Fourier spectra. Given a set of examples X of a function f, the `ComputeFourierSpectrum` method recursively computes $\tilde{\hat{f}}$.

```
      ComputeFourierSpectrum(X, α, i)
(1)     ComputeFourierSpectrum2(X, α, i, 0)
(2)     ComputeFourierSpectrum2(X, α, i, 1)

      ComputeFourierSpectrum2(X, α, i, s)
(3)     if i = n then
(4)       if containsPositiveExamples(X) then f̃̂(α) ← |X| / |X_0|
(5)       else f̃̂(α) ← −|X| / |X_0|
(6)     else
(7)       α_i ← s
(8)       X' ← X
(9)       if s = 1 then
(10)        for each (x, f(x)) ∈ X'
(11)          if x_i = 1 then (x, f(x)) ← (x, −f(x))
(12)      for each (x, f(x)) ∈ X'
(13)        if ∃(y, f(y)) ∈ X' such that ∀j ∈ {j|α_j = *} x_j = y_j then
(14)          if f(x) ≠ f(y) then X' ← X' − {(x, f(x)), (y, f(y))}
(15)      ComputeFourierSpectrum(X', α, i + 1)
```

The `ComputeFourierSpectrum2` procedure performs one of two actions. If all of the digits of α are set (if $i = n$), then it will set the value of $\tilde{\hat{f}}(\alpha)$ at lines 4-5. Otherwise, it will perform the same operations as the `CreateChild` procedure of the best-first search algorithm. The i^{th} digit of α is set to s, and the data set is modified to reflect the fact that α_i has been set to s. These modifications include inverting the outputs of examples and removing any conflicting pairs of examples. Finally, the procedure makes a recursive call to `ComputeFourierSpectrum` to continue the computation.

Notice that at lines 4-5 the sign of the coefficient is determined by checking whether the data set contains positive examples. As a result of inverting the signs of the examples and removing pairs of conflicting examples, X will contain either all positive or all negative examples after all digits of α have been set. If they are positive, then the sign of the coefficient is positive; if they are negative, then the sign

is negative. (This will also be true for the best-first search algorithm, so the sign of each coefficient could be inferred from the remaining examples there as well.)

Unlike the best-first search algorithm, this recursive method for computing the Fourier spectrum does not need to store a search frontier in memory. Therefore, its memory bound is far better than that of the best-first search. It will have at most $n+1$ "nodes" in memory, corresponding to the $n+1$ levels of recursion. Its memory complexity is $O(nk)$, where n is the number of inputs and k is the number of training examples. Contrast this with the $O(mk)$, $n \leq m < 2^n$, memory complexity of the best-first search algorithm. Its run time complexity is identical to the worst-case run time complexity of the best-first search algorithm: $O(2^n k \log k)$.

This recursive method for computing the Fourier spectrum will in some cases outperform the Fast Walsh Transform (FWT). If n is large, then the FWT becomes infeasible since it requires an array of size 2^n to store the entire function (including unknown data points). This recursive algorithm, on the other hand, uses only as much memory as needed to store the known training examples, so it can be used to compute Fourier spectra when the FWT becomes infeasible. Also, unlike the FWT, this recursive algorithm avoids calculations that would involve unknown data points, so it can be beneficial when k is small. When n is small or k is large, however, the FWT, which is more computationally efficient under those conditions, would be preferred.

3.4 Conclusion

The best-first search algorithm presented in this chapter searches through the exponentially-large space of Fourier basis functions to find those that are most highly correlated with a set of training examples. Although there is no guarantee that it will find the most highly correlated functions in polynomial time or space, the results of experiments with several real-world problems have been encouraging. For each real-world problem encountered thus far, only a fraction of the search space has needed to be explored.

Although the best-first search algorithm has proven effective at finding correlated basis functions for each of several data sets, the possibility remains of encountering learning problems for which the search becomes prohibitively difficult. Consequently, an interesting direction of future research will be to analyze the conditions under which the algorithm performs well and under which it performs poorly.

Chapter 4

Real-World Fourier-Based Learning

The best-first search algorithm presented in Chapter 3 demonstrates that it is possible to quickly find the Fourier basis functions that are most highly correlated with the data of a real-world learning problem. Furthermore, the algorithm does not require membership queries. This result opens the door for widely-applicable, real-world Fourier-based learning algorithms.

This chapter presents results of learning real-world problems using methods based on the best-first search algorithm. It also presents an analysis of the Fourier representation that suggests why Fourier-based algorithms may not be well-suited to typical real-world learning problems.

4.1 Fourier-based Learning Methods

This section presents results of applying two Fourier-based learning methods to real-world problems. Both methods are based on the best-first search algorithm of Chapter 3, and both methods output a hypothesis that is a linear combination of Fourier basis functions. The first, and more straightforward, method uses the best-first search to find the r most highly correlated basis functions (the basis functions with the largest coefficients). The second method uses a boosting approach that finds correlated basis functions one-at-a-time, updating a weight distribution on the examples so that subsequent basis functions are more highly correlated with examples that were misclassified by previously selected basis functions.

4.1.1 Learning by Approximating Large Coefficients

The idea of learning by approximating the large coefficients of a function's Fourier representation is the basis of both the LMN and KM algorithms, and research in the learning theory community has shown that functions in several interesting classes can be learned well by approximating the large coefficients of their Fourier representations (see Chapter 2). However, the question of whether real-world problems can effectively be solved by approximating the largest coefficients has not previously been explored. This section presents results of learning real-world problems by approximating the largest Fourier coefficients.

Without an algorithm for efficiently determining which basis functions have the largest coefficients, an algorithm for learning by approximating large coefficients would be infeasible for all but relatively small problems. However, given the best-first search algorithm of the previous chapter, the process of learning by approximating large coefficients is fairly straightforward. In the learning phase, available training data is used to find and approximate the largest coefficients, and the corresponding basis functions are combined to form the hypothesis. Classification of new instances is done by taking a weighted vote of the selected basis functions, where the weights are provided by the basis functions' coefficients.

The only modification made here to this standard approach is to use gradient descent to refine the coefficients of the selected basis functions. This can be thought of as learning "optimal" coefficients for the subset of selected functions. A reasonable argument for such an approach is that since only a subset of the basis functions are being used, the approximated coefficients may not be best. This addition has improved performance in some cases.

Table 4.1 shows the results of learning real-world problems by approximating the largest coefficients. The accuracies reported are the average test accuracies when performing 10-fold cross validation. In each case, the best-first search algorithm was used to find up to 1,000 of the most highly correlated basis functions, and the coefficients of the basis functions were learned by gradient descent. For comparison, results of applying an implementation of the C4.5 decision tree learning algorithm to

the same data sets are also shown.

Table 4.1: Test accuracy of a Fourier-based learner using gradient descent to optimize the coefficients of the most highly correlated basis functions compared to C4.5. Accuracies are the average test accuracy of 30 10-fold cross validation trials.

Data set	Fourier Learner	C4.5 Decision Tree
Chess	85.9	99.4
German	71.5	73.4
Heart	79.9	81.5
Pima	73.6	74.5
SPECT	79.4	80.9
Voting	96.3	96.6
Wisc 1	94.6	94.6
Wisc 2	71.0	75.8
Wisc 3	91.5	94.4

Two interesting results stand out in Table 4.1. First, the results demonstrate that the Fourier-based algorithm generally does a good job learning these problems. The second interesting result, however, is that the Fourier-based algorithm is consistently outperformed by C4.5. Although the Fourier-based learner frequently performs comparably, it never performs better, and in a few cases it performs significantly worse.

These mixed results are both encouraging and discouraging. On the one hand, although the Fourier-based learning algorithm was consistently beaten by C4.5, the learning results are fairly good. Through future work it may be possible to determine how these results can be improved so that Fourier-based learning algorithms perform as well as other learning algorithms. (Chapter 5 shows a generalization of this approach that does compare favorably to C4.5. However, that algorithm is only inspired by this Fourier-based algorithm, and no longer represents a purely Fourier-based approach.)

A discouraging possibility is that despite success in the theoretical realm, pure Fourier-based algorithms may simply be less well-suited to real-world problems than

existing algorithms. Section 4.2 presents arguments and empirical results that seem to support this theory.

4.1.2 Boosting with Fourier Basis Functions

The only Fourier-based learning algorithm that departs significantly from the idea of approximating the largest coefficients is Jackson's Harmonic Sieve (HS) algorithm. The HS algorithm still finds basis functions with large coefficients, but only the coefficient of the first basis function is certain to be large with respect to the original distribution of examples. After finding each basis function, the HS algorithm updates the distribution of examples according to a boosting algorithm so that it focuses more on misclassified examples when finding the next basis function. Thus, the HS algorithm finds basis functions that are well correlated with distributions of examples that are increasingly different from the original distribution.

The HS algorithm essentially uses the KM algorithm to find large coefficients, and therefore requires membership queries. The HS algorithm also uses a boosting technique that is not readily applicable to problems for which oracle queries are not possible. However, by replacing the KM algorithm with the best-first search algorithm of Chapter 3, and by using a boosting method that can be applied to fixed sets of examples, a Fourier-based algorithm that uses boosting can be applied to a more general class of problems.

The boosting algorithm used to obtain the following results is the AdaBoost algorithm of Freund and Schapire [8]. The AdaBoost algorithm assigns a weight to each example that indicates its probability of being sampled. If a learning algorithm is capable of assigning real-valued weights to examples, then these weights can be used directly; otherwise, the data set can be sampled with replacement according to the probability distribution to get a data set that approximates the desired distribution. The best-first search algorithm cannot handle real-valued weights on the examples, so the sampling method is used. (Although the best-first search algorithm cannot handle real-valued weights, it can handle integer-valued weights. Therefore, an alternative could be to approximate the distribution with integer-valued weights.)

The AdaBoost algorithm is used to provide the probability distribution from which sampled data sets are created. The best-first search algorithm is used to find the basis function that is most highly correlated with the current sample. After enough basis functions have been added, the selected basis functions form the ensemble of weak hypotheses that are combined to create a strong hypothesis. The weight of each basis function is provided by the AdaBoost algorithm, which assigns weights to weak hypotheses based on how well they classify their sample of examples.

Unfortunately, the results obtained by attempting this method have generally not been favorable. Table 4.2 shows the results of applying this boosting-based Fourier algorithm to the standard Fourier-based algorithm presented in Section 4.1.1. The accuracies presented indicate the average 10-fold cross validation test accuracy.

Table 4.2: Test accuracy of the boosting-based Fourier learner compared to the standard Fourier learner. Accuracies are the average test accuracy of 10 10-fold cross validation trials.

Data set	Standard Fourier	Boosting Fourier
Chess	85.9	95.3
German	71.5	70.6
Heart	79.9	74.7
Pima	73.6	73.5
SPECT	79.4	78.5
Voting	96.3	96.3
Wisc 1	94.6	91.4
Wisc 2	71.0	62.3
Wisc 3	91.5	90.2

As can be seen in Table 4.2, for many of the problems the boosting-based learner performs worse than the standard Fourier-based learner. The notable exception is the Chess problem. On the Chess problem, the boosting-based learner achieved nearly a 10% absolute increase in accuracy. The exact cause of the generally poor performance is not yet known, and will be an area of future work. For many of the problems, the training set accuracy increases as more basis functions are added, but

unfortunately the generalization accuracy often does not enjoy a similar increase in accuracy.

Another problem with the boosting-based learner is related to the best-first search. As shown in Chapter 3, the best-first search algorithm is able to quickly find thousands of the most highly correlated basis functions for each of the data sets. However, as the boosting algorithm alters the data sets, it becomes increasingly difficult for the algorithm to find even the single most highly correlated function. As a result, the boosting algorithm is sometimes forced to stop prematurely because the search algorithm cannot find the next-best basis function. Although this is not a problem for most of the data sets, for the Chess, Wisc 1, and Wisc 3 data sets it is quite possible that the accuracy could be improved if more basis functions could be added.

Thus, while the best-first search algorithm has excelled on all of the original data sets that it has been tested on, some of the resampled data sets, whose distributions have been skewed by the AdaBoost algorithm, provide examples of data sets that approach worst-case scenarios for the search algorithm. An important area of future work will be to determine if the search algorithm can be modified so that it does not perform poorly when faced with such data sets.

4.2 Analyzing the Fourier Representation

A study of the real-world usefulness of Fourier-based learning methods naturally leads to an examination of the utility of the Fourier representation. This section describes some properties of the Fourier representation, and presents arguments and empirical results that suggest why the Fourier representation may not be well suited to typical real-world problems.

4.2.1 The Fourier Representation

Recall that Fourier-based learning algorithms represent functions as a linear combination of Fourier basis functions. As described in Chapter 2, the Fourier basis

functions are parity functions, each computing the parity, or XOR, of a subset of the inputs.

Perhaps the first relevant property of the Fourier representation is that the set of XOR functions over all subsets of inputs forms a basis for the space of Boolean functions. In other words, any Boolean function can be represented as a linear combination of the Fourier basis functions—a valuable property for a learning system. However, because only a relatively small subset of the basis functions are selected by Fourier-based learning algorithms, in practice it will not be possible to represent every function perfectly.

Given that the hypothesis output by a Fourier-based algorithm is a combination of a limited number of XOR functions, it would seem that a Fourier-based algorithm would have an advantage over other learning algorithms when one or more high-order XORs of the inputs were relevant features for learning the task. On the other hand, the Fourier representation would seem less beneficial if high-order XORs do not tend to be useful features for real-world problems. The following section considers the utility of XOR features.

4.2.2 An Argument Against the Fourier Representation

A "no free lunch" argument [22] would suggest that the Fourier representation is as useful as any other representation. When considering all possible functions, there will be just as many functions for which high-order XOR relationships are useful as there will for any other type of relationship. If all functions were equally likely to be encountered in practice, then it would indeed be true that no type of high-order relationship would be more beneficial than any other. However, there is reason to believe that all functions are not equally likely to be encountered in real-world learning scenarios.

A fundamental difference between real-world learning problems and the set of all possible learning problems is that real-world problems are designed by people. Given some task to learn, a person selects features that they believe will be useful for solving the problem. Because people select the features to solve the problem, it seems

reasonable to assume that the relationships between those features that are relevant for solving the problem will be indicative of human thought processes.

People seem to think very naturally in terms of conjunctions (ANDs) and disjunctions (ORs). In other words, when selecting features for solving a problem, a person could naturally think, "If each of these are true...", or "If any of these are true...". These phrases exemplify AND and OR relationships, respectively. Because people tend to think naturally in terms of AND and OR, it seems reasonable to expect that useful information may be found in AND or OR combinations of the inputs of a learning problem.

On the hand, consider the parity, or XOR, relationship. A high-order XOR relationship would be exemplified by a thought process such as, "If an odd number of these are true...". Notice that it does not matter which inputs are true, or even how many are true, only that an odd number are true. It seems unlikely that a person would select features with such a relationship in mind. The 2^{nd}-order XOR relationship is slightly more intuitive: "If either one but not both of these are true...". However, even this relationship seems far less likely than the more intuitive AND and OR relationships.

To test the prevalence of AND, OR, and XOR relationships in real-world data, each of the data sets used in this thesis was searched for high-order AND, OR, and XOR features that are well correlated with the output. These high-order AND, OR, and XOR features compute the logical AND, OR, and XOR, respectively, of two or more of the original input features. (Notice that every high-order XOR feature is computed by a Fourier basis function. Because the Fourier transform allows for negative coefficients, the Fourier basis functions can be inverted, and therefore the Fourier basis functions also account for all XNOR relationships. Patterning AND and OR functions after the Fourier basis functions, the AND and OR functions account for all NAND and NOR relationships. For simplicity, the following discussion will refer only to AND, OR, and XOR relationships. Keep in mind, however, that any reference to an AND, OR, or XOR relationship could refer to a NAND, NOR, or XNOR relationship.)

For each type of high-order Boolean feature, the classification accuracy of the subset of two or more inputs that was most highly correlated with the output was recorded. In other words, if the logical OR of inputs 2, 3, and 5 was the most highly correlated OR feature, then the accuracy with which the OR of those inputs classified each example was recorded. This experiment was carried out for all three types of relationships on each data set. In addition to the 2^{nd}- and higher-order Boolean relationships, the accuracy of the best 1^{st}-order feature for each data set was also recorded. (The 1^{st}-order features are the original input features; therefore, the accuracy of a 1^{st}-order feature is the accuracy with which an individual input predicts the output by itself.) The results of this experiment are shown in Table 4.3. The best accuracy for each data set is highlighted in bold.

Table 4.3: Best classification accuracy of any 1^{st}-order features and any higher-order AND, OR, or XOR features. For each data set, the best accuracy is highlighted in bold.

Data set	1^{st}	AND	OR	XOR
Chess	68.3	67.7	**81.1**	75.3
German	71.7	71.7	**73.1**	71.7
Heart	75.6	76.3	**77.0**	76.3
Pima	73.6	**75.4**	71.1	65.9
SPECT	66.3	79.4	**87.6**	70.8
Voting	**96.3**	95.9	90.1	88.1
Wisc 1	87.3	87.1	**96.0**	92.7
Wisc 2	76.8	**80.3**	78.8	77.3
Wisc 3	91.4	**94.4**	89.8	91.2

Interestingly, the most highly correlated high-order feature was always either an AND or an OR feature. For the Voting data set, there was a first-order feature that was better than any high-order AND, OR, or XOR feature, but of the three types of high-order features an AND feature was best. The best XOR feature was sometimes not far behind the best AND or OR feature, and it was not always the worst of the three, but it was never the best. This result seems significant given that

there was no prior reason, other than the arguments presented previously, to believe that any type of high-order Boolean feature would be more or less useful than the others for these data sets.

These results suggest that AND and OR relationships may be more relevant to real-world learning problems than XOR relationships. If this is the case, learning algorithms that learn AND and OR relationships would seem more likely to be successful than algorithms that learn by examining XOR relationships. Consequently, Fourier-based algorithms, which rely on XOR relationships, may be a poor choice for typical real-world problems.

Although the arguments and empirical results of this section suggest that high-order AND and OR features are better than XOR features for learning real-world problems, they do not rule out the existence of real-world problems for which XORs are better than ANDs and ORs. An interesting area for future work will be to search for such problems, and to analyze the properties of a learning problem that might cause XORs to be more useful.

4.3 Conclusion

The results of learning real-world problems with Fourier-based techniques based on the best-first search for coefficients are mixed. On the one hand, the Fourier-based learning algorithms are able to achieve reasonably good learning accuracies. On the other hand, the Fourier-based algorithms struggle to perform as well as existing learning algorithms, as demonstrated in the comparison with C4.5. Furthermore, the analysis of Section 4.2 suggests that the mediocre results may be due to an inherent weakness (with respect to real-world problems) in the Fourier representation.

Based on these results, it would seem that Fourier-based learning algorithms will generally not be as useful in practice as they have been in theory. However, it may be that the sub-optimal results are a consequence of the techniques used, and not necessarily a fundamental weakness of Fourier-based methods. Consequently, an interesting area for future work will be to determine if alternative Fourier-based methods can achieve better learning results.

Finally, one positive result of the otherwise negative analysis in Section 4.2, is the development of a generalization of these Fourier techniques that significantly improves learning accuracy. The following chapter presents an algorithm that learns by finding highly correlated functions from any set of functions defined by a Boolean operator. This includes the set of XOR functions, but also includes other sets of functions, such as the sets of AND and OR functions.

Chapter 5

Generalized Fourier-Based Learning

The previous chapter presented analysis and empirical results that suggest that the Fourier basis may provide a poor representation for the learning problems likely to be encountered in practice. In particular, the XOR functions of the Fourier basis seem less likely to be useful than the sets of AND and OR functions. Conveniently, the best-first search algorithm presented in Chapter 3 can be generalized to accommodate any well-defined "basis" of functions, including the sets of all AND and OR functions.

By generalizing the best-first search algorithm, it is possible to learn functions via any basis of functions. Furthermore, these bases can be combined, as the search algorithm can readily explore multiple bases simultaneously. This chapter demonstrates how the best-first search can be generalized, and it presents results of learning real-world problems with a combination of bases.

5.1 Definitions and Notation

In addition to the XOR functions of the Fourier basis, this chapter will also examine AND and OR functions, which compute the logical AND and OR, respectively, of subsets of inputs. The AND, OR, and XOR functions are defined as follows:

$$AND_\alpha(x) = \begin{cases} 1 : \text{if } \sum_{i=0}^{n-1} \alpha_i x_i < \sum_{i=0}^{n-1} \alpha_i \\ -1 : \text{if } \sum_{i=0}^{n-1} \alpha_i x_i = \sum_{i=0}^{n-1} \alpha_i \end{cases}$$

$$OR_\alpha(x) = \begin{cases} 1 : \text{if } \sum_{i=0}^{n-1} \alpha_i x_i = 0 \\ -1 : \text{if } \sum_{i=0}^{n-1} \alpha_i x_i > 0 \end{cases}$$

$$XOR_\alpha(x) = \begin{cases} 1 \ : \text{ if } \ \sum_{i=0}^{n-1} \alpha_i x_i \text{ is even} \\ -1 \ : \text{ if } \ \sum_{i=0}^{n-1} \alpha_i x_i \text{ is odd} \end{cases}$$

where $\alpha \in \{0,1\}^n$.

As with the Fourier basis functions, these functions can also be defined in terms of the subset of inputs used in their computation. Recall that the inputs that are used are those for which $\alpha_i = 1$. Let $S \subseteq \{0, 1, ..., n-1\}$ be the set of all i such that $\alpha_i = 1$. Then the AND, OR, and XOR functions can be defined as follows:

$$AND_S(x) = \begin{cases} 1 \ : \text{ if } \ \sum_{i \in S} x_i < |S| \\ -1 \ : \text{ if } \ \sum_{i \in S} x_i = |S| \end{cases}$$

$$OR_S(x) = \begin{cases} 1 \ : \text{ if } \ \sum_{i \in S} x_i = 0 \\ -1 \ : \text{ if } \ \sum_{i \in S} x_i > 0 \end{cases}$$

$$XOR_S(x) = \begin{cases} 1 \ : \text{ if } \ \sum_{i \in S} x_i \text{ is even} \\ -1 \ : \text{ if } \ \sum_{i \in S} x_i \text{ is odd} \end{cases}$$

Notice that the definitions of the XOR functions are identical to the definitions of the Fourier basis functions given in Chapter 2 (see Equations 2.2 and 2.3).

Coefficients can be computed for these sets of functions in the same manner as for the Fourier basis functions simply by replacing χ_α with the appropriate function type:

$$\tilde{\hat{f}}_{AND}(\alpha) = \frac{1}{|X|} \sum_{x \in X} f(x) AND_\alpha(x)$$

$$\tilde{\hat{f}}_{OR}(\alpha) = \frac{1}{|X|} \sum_{x \in X} f(x) OR_\alpha(x)$$

$$\tilde{\hat{f}}_{XOR}(\alpha) = \frac{1}{|X|} \sum_{x \in X} f(x) XOR_\alpha(x)$$

In the case of the AND and OR functions, these coefficients do not necessarily yield a linear combination that fits the data, but the coefficients do indicate the correlation between each function and the data.

5.2 Generalizing the Best-First Search

The best-first search algorithm can be generalized for other bases of functions quite easily. The `FindLargestCoefficients` procedure that was presented previously (Figure 3.2) already describes the general technique for finding correlated functions in any basis. The procedure is repeated in Figure 5.1.

Figure 5.1: A best-first first search algorithm for finding the most highly correlated functions of any basis. The `FindLargestCoefficients` procedure takes as input a set of examples X, a number of basis functions to find r, and returns R, the set of labels of the r most highly correlated basis functions.

```
      FindLargestCoefficients(X, r)
(1)     rootNode.β ← *^n
(2)     rootNode.X_β ← X
(3)     priorityQueue.Insert(rootNode)
(4)     R ← {}
(5)     while |R| < r
(6)       currentNode ← priorityQueue.RemoveFront()
(7)       if ∀i (currentNode.β_i ≠ *) then R ← R ∪ currentNode.β
(8)       else
(9)         i ← SelectDigit(currentNode)
(10)        priorityQueue.Insert(CreateChild(currentNode, i, 0))
(11)        priorityQueue.Insert(CreateChild(currentNode, i, 1))
(12)    return R
```

The only significant changes that must be made to the best-first search algorithm are in the `CreateChild` procedure. The primary function of this procedure is to identify and remove pairs of examples that are conflicting with respect to the region of the search space that a child node represents. Thus, adapting the search algorithm to different bases of functions amounts primarily to altering the `CreateChild` procedure so that it correctly removes examples that are conflicting with respect to specific regions of the desired basis.

Only relatively small changes must be made to the `CreateChild` procedure to allow it handle the AND and OR bases. Figure 5.2 shows a `CreateChild` procedure

that can be used with the AND, OR, and XOR bases.

Figure 5.2: The `CreateChild` procedure of the generalized best-first search algorithm. It takes as input a node of the best-first search, an index i, and a value s, and returns the child node that results from setting β_i to s.

```
      CreateChild(parent, i, s)
(1)     child.X_β ← parent.X_β
(2)     child.β ← parent.β
(3)     child.β_i ← s
(4)     child.count_β ← parent.count_β

(5)     if s = 1 then
(6)       if isANDNode(parent) then
(7)         for each (x, f(x)) ∈ child.X_β
(8)           if x_i = 0 then
(9)             child.count_β ← child.count_β − f(x)
(10)            child.X_β ← child.X_β − {(x, f(x))}
(11)      if isORNode(parent) then
(12)        for each (x, f(x)) ∈ child.X_β
(13)          if x_i = 1 then
(14)            child.count_β ← child.count_β − f(x)
(15)            child.X_β ← child.X_β − {(x, f(x))}
(16)      if isXORNode(parent) then
(17)        for each (x, f(x)) ∈ child.X_β
(18)          if x_i = 1 then (x, f(x)) ← (x, −f(x))

(19)    for each (x, f(x)) ∈ child.X_β
(20)      if ∃(y, f(y)) ∈ child.X_β such that ∀j ∈ {j|β_j = ∗} x_j = y_j then
(21)        if f(x) ≠ f(y) then child.X_β ← child.X_β − {(x, f(x)), (y, f(y))}
(22)    return child
```

The basic structure of this `CreateChild` procedure is unchanged from the version presented in Chapter 3. At the beginning of the procedure (lines 1-4), the child node is initialized, and at the end (lines 19-22), any conflicting examples are removed. The steps for removing conflicting examples are the same here as in the previous version of the procedure. The only change at the beginning of the procedure is the initialization of a new variable, $count_\beta$ (line 4), that is used for the AND and

OR bases.

The most significant change to the `CreateChild` procedure is in the midsection of the procedure (lines 5-18). This section is only executed when $s = 1$, indicating that the i^{th} input is to be used by functions in this node's region. If $s = 1$, then different actions are taken depending on the type of function that the node represents. The last portion of this section (lines 16-18) is for the XOR basis, and the actions taken are the same as in the `CreateChild` procedure introduced in Chapter 3. Lines 6-10 and lines 11-15 handle the AND and OR bases, respectively.

If the node represents a region of the AND basis (line 6), then the data set is searched for examples such that $x_i = 0$ (lines 7-8). For each such example found, the node's $count_\beta$ variable is updated by subtracting the example's output from its current value (line 9), after which the example is removed from the data set (line 10). Similar actions are taken when the node represents a region of the OR basis (lines 14-15). In that case, however, the actions are taken if $x_i = 1$ (line 13).

The key to these methods is the observation that the output of an AND or OR function can be determined by finding a single input that decides the output. For AND functions, having one input equal to 0 makes the values of the other inputs irrelevant. For OR functions, the same is true when one input has the value 1. Thus, after finding such an input in an example, it can be removed, and its effect on the coefficient can be taken into account.

The effect on the coefficient of examples whose classifications have been determined is accounted for by a node's $count_\beta$ variable. For AND (OR) nodes, $count_\beta$ keeps a running total of the number of negative examples that will be classified as 1 (-1) minus the number of positive examples that will classified as 1 (-1). More importantly, this value provides a running total of the number of non-conflicting examples whose classifications have been determined. Notice that for every pair of such examples, if one is positive and the other is negative then $count_\beta$ will increase by 1 and decrease by 1; in other words, the two examples will cancel each other out. Since the examples will be classified identically by functions in this region but have different outputs, they are conflicting. By canceling each other out, they do not add to the

absolute value of $count_\beta$. Since conflicting examples cancel each other out in this way, the absolute value of $count_\beta$ gives the number of examples whose classifications have been determined that do not belong to a conflicting pair.

Now, instead of being bounded by the number of training examples at a node, the largest possible coefficient in any node's region is bounded by the number of training examples at that node plus the absolute value of $count_\beta$:

$$\max_{\alpha \in \beta} \tilde{\hat{f}}(\alpha) \leq \frac{|X_\beta| + abs(count_\beta)}{|X|}$$

Notice that $|X_\beta|$ tells how many of the examples whose classifications have not yet been determined are non-conflicting, while $count_\beta$ tells how many of the examples whose classifications have already been determined are non-conflicting. (Although $count_\beta$ is only required for AND and OR bases, the above inequality is correct for the XOR basis as well. For the XOR basis, $count_\beta$ will always be 0, reducing the inequality to the one introduced for the Fourier basis in Equation 3.4.)

At a leaf node, the classifications of all examples are known. For AND nodes, the only remaining examples will be those for which $x_i = 1$ for all i such that $\alpha_i = 1$ (all other examples would have been removed and had their outputs subtracted from $count_\beta$). Therefore, the remaining examples will all be classified as -1. (Recall that by definition a positive output is denoted by -1, while a negative output is denoted by 1.) Similarly, for OR nodes, the remaining examples will be those for which $x_i = 0$ for all i such that $\alpha_i = 1$. Those examples will all be classified as 1. Since the classifications of these examples are known, they can be compared with $count_\beta$ to identify more potential conflicts. Because the remaining examples will be classified differently than the examples that have previously been removed, they will be conflicting if their outputs are the same as the outputs of the examples that are implicitly represented by $count_\beta$. The following formula, which computes the absolute value of the coefficient, takes this into account directly:

$$\tilde{\hat{f}}(\alpha) = \frac{abs(count_\alpha \pm |X_\alpha|)}{|X|}$$

In the above formula, $|X_\alpha|$ is added to $count_\beta$ if X_α contains positive examples, and is subtracted from it otherwise.

The previous discussion demonstrates how the search algorithm can be modified to explore alternative bases of functions. In addition to being able to explore different bases of functions, the algorithm can search through multiple bases simultaneously. To explore multiple bases simultaneously, line 3 of `FindLargestCoefficients` must be altered so that the priority queue is initialized with a root node for each basis. Then, the search algorithm will automatically explore the bases simultaneously. Conceptually, the algorithm can be thought of as jumping between the three search trees, always exploring the region that appears most promising.

5.3 Generalized Basis Function Learning

By generalizing the best-first search so that other bases of functions can be used, it is possible to have access to a richer set of basis functions when learning. If a particular basis of functions is believed to be particularly relevant to a given learning task, that basis can be used by the learner. Or, prior assumptions can be ignored, as the search algorithm can explore multiple bases simultaneously to discover which basis contains well-correlated basis functions. Furthermore, functions from multiple bases can be combined in a heterogeneous hypothesis.

The analysis and experiments described in the previous chapter suggest that AND and OR functions might be more useful for typical learning real-world problems than XOR functions. This section describes the generalized basis function learning method, and presents results of learning the same problems used in the previous chapter after adding the AND and OR bases to the algorithm.

5.3.1 Learning with Arbitrary Bases

Given a generalized search algorithm, generalizing the learning method to use arbitrary bases is straightforward. Let A denote the set of labels of the most highly correlated functions returned by the search algorithm, and let B_α, $\alpha \in A$, denote the basis function with label α. These basis functions may all belong to the same basis or be drawn from different bases. (If A may contain functions from different bases, each label α must also indicate the basis to which the function belongs.) The

generalized basis function classifier c is simply a thresholded linear combination of the most highly correlated basis functions:

$$c(x) = sgn\left(\sum_{\alpha \in A} \hat{f}(\alpha) B_\alpha(x)\right)$$

where sgn is defined as before:

$$sgn(x) = \begin{cases} 1 : \text{if } x \geq 0 \\ -1 : \text{if } x < 0 \end{cases}$$

5.3.2 Results

Adding the AND and OR bases to the standard Fourier (XOR) basis increases learning accuracy significantly for several of the learning tasks. Figure 5.1 shows a comparison of learning accuracy when using the Fourier basis and when using the AND, OR, and XOR bases. Each accuracy is the average 10-fold cross validation test accuracy over 30 trials. Where there was a statistically significant difference in accuracy, the higher accuracy is highlighted in bold.

Table 5.1: A comparison of learning accuracy when using the XOR (Fourier) basis and when using a combination of AND, OR, and XOR bases. The accuracies are the average test accuracy of 30 10-fold cross validations. Where there is a statistically significant difference in accuracy, the higher accuracy is highlighted in bold.

Data set	XOR	AND/OR/XOR
Chess	**85.9**	81.0
German	71.5	71.5
Heart	79.9	**84.4**
Pima	73.6	**76.1**
SPECT	79.4	**84.0**
Voting	96.3	96.3
Wisc 1	94.6	**95.1**
Wisc 2	71.0	**74.9**
Wisc 3	91.5	**93.6**

For most of the data sets, a statistically significant increase in accuracy was achieved by adding the AND and OR bases to the XOR basis of the Fourier transform.

Interestingly, for the Chess problem, the learning accuracy decreased when access to all three bases was given. Preliminary investigations into this issue suggest that the problem may be a result of the most highly correlated basis functions being too correlated with each other.

When using the AND, OR, and XOR bases, the learning accuracy compared much more favorably with C4.5. Figure 5.2 shows this comparison. For each data set, the higher accuracy is highlighted in bold.

Table 5.2: The test accuracy of the generalized AND/OR/XOR basis function learner compared to the C4.5 decision tree algorithm. The accuracies are the average of 30 10-fold cross validations. For each data set, the higher accuracy is highlighted in bold.

Data set	AND/OR/XOR	C4.5
Chess	81.0	**99.4**
German	71.5	**73.4**
Heart	**84.4**	81.5
Pima	**76.1**	74.5
SPECT	**84.0**	80.9
Voting	96.3	**96.6**
Wisc 1	**95.1**	94.6
Wisc 2	74.9	**75.8**
Wisc 3	93.6	**94.4**

5.3.3 Analysis

In addition to providing good accuracies on several data sets, this learning approach has several useful properties. For example, it is capable of learning complex functions while remaining relatively comprehensible. Contrast the basis functions returned by the search algorithm (the AND of inputs 3 and 4, the XOR of inputs 5, 8, and 9, etc.) with the complex high-order features learned in the hidden layers of a neural network, for example.

Another useful property of the algorithm is its ability to find several types of correlations. Some learning algorithms learn a particular type of correlation well,

but struggle to learn others. For example, an algorithm may learn conjunctions of attributes quite well, but struggle to learn XOR relationships. The algorithm presented here is capable of finding and using AND, OR, and XOR correlations. In addition, if deemed useful, other types of correlations could be added to the search.

5.4 Additional Applications

In addition to the learning algorithm that has already been presented, there are other potential uses for the best-first search algorithm. One such use is data analysis. Every coefficient gives information about the correlation between some subset of the inputs and the output. By applying the best-first search algorithm to search for useful correlations, interesting properties of the data can be efficiently determined. For example, running the algorithm to find correlated basis functions for the SPECT data set reveals an OR function with a strong correlation to the output (much stronger than any AND or XOR correlations). This particular basis function computes the logical OR of eight inputs. Such a high-order correlation would be nearly impossible for humans to observe, but the search algorithm finds it in seconds.

The search algorithm also has potential benefits as an automatic feature selector. The search for basis functions can be thought of as a search for features that are relevant to the learning task. These features can be used as inputs to any existing learning algorithm. This technique may be especially useful to learning algorithms that don't easily learn high-order features. Consider again the example of the high-order OR correlation found in the SPECT data set. The absolute accuracy of the perceptron learning algorithm is increased by 2% by adding the high-order OR function as an additional feature.

5.5 Conclusion

Although the best-first search algorithm was designed to provide an efficient means of finding the most highly correlated Fourier basis functions, the fact that it generalizes to allow searches through combinations of arbitrary bases may be more

interesting. The improvement in learning accuracy obtained by providing access to additional bases demonstrates the potential benefits of this generalized approach.

The ability to use arbitrary bases of functions introduces several interesting questions. For example, what other bases of functions might be useful for solving learning problems? Can an appropriate choice of basis be determined by examining properties of the learning problem or training data? Which bases of functions are useful to use in combination? Which bases are largely redundant? Future research will attempt to answer these questions.

Chapter 6

Conclusion

The primary result of this thesis is the best-first search algorithm for finding the basis functions that are most highly correlated with real-world data sets. Although the space of basis functions is exponentially large, experiments with real-world problems have shown that only a small fraction of the space needs to be explored to find those that are most highly correlated. Furthermore, the algorithm runs very quickly, making it practical for real-world usage in cases where the Fast Walsh Transform is not feasible.

In additional to its efficiency, an important aspect of the algorithm is its general applicability. Most importantly, the algorithm is designed to learn from fixed sets of data and does not need to query an oracle for examples during its training phase.

The ability to quickly find the most correlated basis functions opens the door for practical real-world Fourier-based learning algorithms. Testing the Fourier-based learning method on several real-world problems reveals that they can be learned well, although the results are not particularly impressive.

An evaluation of the Fourier representation suggests that the Fourier basis may not be ideal for typical real-world problems. The fact that real-world learning problems are biased by a human-driven feature selection process suggests that the relationships between input features that are likely to be useful for learning are those that are indicative of human thought processes. Consequently, the XOR relationships of the Fourier basis seem less likely to be useful than more intuitive AND and OR relationships. Empirical results seem to support this claim, as for each data set the best XOR features never classified the examples as accurately as either the best AND

or best OR features.

Finally, the negative analysis of the Fourier representation is alleviated by a positive result. The best-first search algorithm for finding correlated basis functions generalizes to allow searches through arbitrary "bases" of functions, and can even search through multiple bases simultaneously. This allows the algorithm to search through a variety of interesting bases to find the basis or combination of bases that is best for a particular problem. Empirical results reveal that learning accuracy increases significantly when the Fourier-based learner is expanded to include bases of AND and OR functions in its search.

Although the best-first search algorithm has proven effective at finding the most correlated basis functions, there are no guarantees for future performance. A beneficial area of future research would be to provide precise descriptions of the conditions under which the algorithm performs well. Once these conditions have been identified, another area for future work would be to determine if the search algorithm can be modified so that the difficult cases can be handled more efficiently.

Other areas for future work include identifying ways to improve the lackluster results of learning with the Fourier basis. In particular, can the mixed results of the boosting-based approach be improved? Further consideration might also be given to the suggested weaknesses of the Fourier basis. Can interesting real-world problems be found for which the Fourier basis outperforms other bases? If so, what are the characteristics of those learning problems?

Finally, there area many open questions regarding the generalized basis function learning methods. For example, what alternative bases of functions might be useful? How do the representational power of the different bases compare with one another? Which bases are more useful for their theoretical properties, and which are more useful in practical application? Finally, all of these issues can be further explored in the context of combinations of bases.

Appendix A

Data Sets

The data sets used in this thesis, summarized in Table A.1, can be found in the UCI Machine Learning Repository [21]. Each data set represents a Boolean classification problem. Several of the data sets contained non-Boolean input features that needed to be transformed into Boolean features for compatability. Table A.1 shows how many of the original input features were Boolean, continuous, or nominal (having three or more non-numerical values).

Table A.1: A summary of the data sets used in this thesis, showing the original number of inputs, the number of inputs after Boolean encoding, and the number of examples. In parentheses, the original number of inputs is broken down by input type: Boolean, continuous, or nominal.

Data set	# Inputs (Bool, Cont, Nom)	# Inputs After Encoding	# Examples
Chess	36 (35, 0, 1)	37	3196
German	24 (12, 12, 0)	24	1000
Heart	13 (3, 7, 3)	16	270
Pima	8 (8, 0, 0)	8	768
SPECT	22 (22, 0, 0)	22	267
Voting	16 (16, 0, 0)	16	435
Wisc 1	9 (0, 0, 9)	36	699
Wisc 2	33 (0, 33, 0)	33	198
Wisc 3	30 (0, 30, 0)	30	569

Each continuous input was encoded as a single Boolean input by finding a threshold that maximized information gain and labeling input values above and below

the threshold as positive and negative, repsectively. Each nominal input was encoded as a sequence of n Boolean inputs, where n was the minimum number of bits required to assign each possible value a unique binary encoding.

Bibliography

[1] A. Blum, M. Furst, J. Jackson, M. Kearns, Y. Mansour, and S. 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.

[2] N. Bshouty and V. Feldman. On using extended statistical queries to avoid membership queries. *Journal of Machine Learning Research*, 2:359–395, 2002.

[3] N. Bshouty, J. Jackson, and C. Tamon. More efficient PAC-learning of DNF with membership queries under the uniform distribution. *Journal of Computer and System Sciences*, 68:205–234, 2004.

[4] N. Bshouty and C. Tamon. On the Fourier spectrum of monotone functions. *Journal of the ACM*, 43:747–770, 1996.

[5] A. Drake and D. Ventura. Comparing high-order boolean features. In *Proceedings of the 8th Joint Conference on Information Sciences*, pages 428–431, 2005.

[6] A. Drake and D. Ventura. A practical generalization of Fourier-based learning. In *Proceedings of the 22nd International Conference on Machine Learning*, pages 185–192, 2005.

[7] Y. Freund. Boosting a weak learning algorithm by majority. In *Proceedings of the 3rd Annual Workshop on Computational Learning Theory*, pages 202–216, 1990.

[8] Y. Freund and R. Schapire. Experiments with a new boosting algorithm. In *Proceedings of the 13th International Conference on Machine Learning*, volume 55, pages 148–156, 1996.

[9] O. Goldreich and L. Levin. A hard-core predicate for all one-way functions. In *Proceedings of the 21st ACM Symposium on Theory of Computing*, pages 25–32, 1989.

[10] S. L. Hurst, D. M. Miller, and J. C. Muzio. *Spectral Techniques in Digital Logic*. Academic Press, Inc., 1985.

[11] J. Jackson. An efficient membership-query algorithm for learning DNF with respect to the uniform distribution. *Journal of Computer and System Sciences*, 55:414–440, 1997.

[12] J. Jackson, A. Klivans, and R. Servedio. Learnability beyond AC^0. In *Proceedings on 34th Annual ACM Symposium on Theory of Computing*, pages 776–784, 2002.

[13] Roni Khardon. On using the Fourier transform to learn disjoint DNF. *Information Processing Letters*, 49:219–222, 1994.

[14] A. Klivans, R. O'Donnell, and R. Servedio. Learning intersections and thresholds of halfspaces. *Journal of Computer and System Sciences*, 68:808–840, 2004.

[15] A. Klivans and R. Servedio. Boosting and hard-core sets. In *40th Annual IEEE Symposium on Foundations of Computer Science*, pages 624–633, 1999.

[16] E. Kushilevitz and Y. Mansour. Learning decision trees using the Fourier spectrum. *SIAM Journal on Computing*, 22(6):1331–1348, 1993.

[17] N. Linial, Y. Mansour, and N. Nisan. Constant depth circuits, Fourier transform, and learnability. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science*, pages 574–579, 1989.

[18] N. Linial, Y. Mansour, and N. Nisan. Constant depth circuits, Fourier transform, and learnability. *Journal of the ACM*, 40(3):607–620, 1993.

[19] Y. Mansour. An $O(n^{\log\log n})$ learning algorithm for DNF under the uniform distribution. *Journal of Computer and System Sciences*, 50:543–550, 1995.

[20] Y. Mansour and S. Sahar. Implementation issues in the Fourier transform algorithm. *Machine Learning*, 14:5–33, 2000.

[21] D.J. Newman, S. Hettich, C.L. Blake, and C.J. Merz. UCI repository of machine learning databases, 1998.

[22] D. Wolpert and W. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computing*, 1(1):67–82, 1997.