2010-06-30

Practical Improvements in Applied Spectral Learning

Adam C. Drake
Brigham Young University - Provo

Practical Improvements in Applied Spectral Learning

Adam Drake

A dissertation submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Dan Ventura, Chair
Tony Martinez
Christophe Giraud-Carrier
Scott Woodfield
Jay McCarthy

Department of Computer Science

Brigham Young University

August 2010

ABSTRACT

Practical Improvements in Applied Spectral Learning

Adam Drake

Department of Computer Science

Doctor of Philosophy

Spectral learning algorithms, which learn an unknown function by learning a spectral representation of the function, have been widely used in computational learning theory to prove many interesting learnability results. These algorithms have also been successfully used in real-world applications. However, previous work has left open many questions about how to best use these methods in real-world learning scenarios.

This dissertation presents several significant advances in real-world spectral learning. It presents new algorithms for finding large spectral coefficients (a key sub-problem in spectral learning) that allow spectral learning methods to be applied to much larger problems and to a wider range of problems than was possible with previous approaches. It presents an empirical comparison of new and existing spectral learning methods, showing among other things that the most common approach seems to be the least effective in typical real-world settings. It also presents a multi-spectrum learning approach in which a learner makes use of multiple representations when training. Empirical results show that a multi-spectrum learner can usually match or exceed the performance of the best single-spectrum learner. Finally, this dissertation shows how a particular application, sentiment analysis, can benefit from a spectral approach, as the standard approach to the problem is significantly improved by incorporating spectral features into the learning process.

Keywords: spectral learning, Fourier-based learning, Fourier transform, machine learning, search algorithms, sentiment analysis

Contents

Chapter 1

Introduction

One of the most significant advances in computational learning theory has been the development of algorithms that learn an unknown function by learning a Fourier representation of the function. These spectral learning algorithms have led to many significant theoretical advances, as they have made it possible for researchers to prove many new learnability results about the classes of functions that can be learned (efficiently) in various scenarios. These spectral learning algorithms have also led to successful real-world applications.

However, many open questions and issues in spectral learning are left unresolved by previous work. For example, many of the algorithms presented in computational learning theory cannot be applied in real-world domains because they rely on assumptions that significantly limit their applicability. In addition, several different approaches to learning spectral representations have been presented, but it is not clear which are most effective in typical real-world learning scenarios. And, although previous work has focused on the Fourier representation, this is just one of many possible representations that a spectral learner might use, and it seems natural to consider how a learner might take advantage of multiple representations. This dissertation explores and addresses these and other open questions and issues.

Chapter 2 is devoted to a central sub-problem of spectral learning: finding the largest coefficients of a function's spectral representation. Spectral learning algorithms typically learn a function by identifying and approximating the largest spectral coefficients. However, the number of coefficients is exponential in the number of inputs to a function, making brute-

force approaches to computing and finding large coefficients impractical for large learning problems. Some previous approaches have avoided this difficulty by relying on assumptions about which coefficients will be large. However, by not considering all coefficients, the representational power of these approaches is limited. Algorithms have been developed that do explore the entire space of coefficients, but they are not widely applicable since they rely on assumptions that are not typical of real-world scenarios (e.g., assuming that specific examples can be requested while training). This chapter focuses on the problem of finding large coefficients in the common scenario in which a learner must learn an unknown function from a fixed set of examples. It first presents a theoretical analysis of the hardness (i.e., NP-completeness) of the problem. Then, it introduces a practical method for bounding the size of the largest possible coefficient in any region of a spectral representation, and shows how the method can be incorporated into search algorithms that can find large spectral coefficients without computing the entire spectrum.

Chapter 3 addresses the question of what is the best way to learn spectral representations. Several different approaches to spectral learning have been presented, but it is not clear from previous work which of these methods are best for real-world learning scenarios. This chapter breaks down the spectral learning process into two sub-problems: (1) determining which basis functions to use, and (2) determining how to set the coefficients of the selected basis functions. It compares different combinations of the most significant previous approaches to these sub-problems, and shows which approaches work best in real-world settings.

Chapter 4 proposes a multi-spectrum approach to spectral learning. In previous work, spectral learning algorithms have used a single representation. However, there are many possible representations that a learner could use, and it is easy to show that no one representation can be best for all learning problems. This chapter suggests three fundamental ways that a spectral learner might use multiple representations when learning, and it shows which of these approaches perform best in real-world settings.

In Chapter 5, the dissertation departs briefly from its spectral learning theme to introduce a specific application to which spectral learning is applied in Chapter 6. The application introduced in Chapter 5 is the problem of learning to assign real-valued scores to documents that indicate the overall positive or negative sentiment being expressed. This "sentiment regression" problem can be viewed as a fine-grained variant of the typical sentiment classification problem of assigning a document to one of a few sentiment categories (e.g., positive, negative, or neutral). This chapter compares the performance of a few machine learning algorithms at this task, and it analyzes the appropriateness of a regression approach (relative to the commonly-used classification approach), as it compares the performance of classification and regression approaches as the number of sentiment categories increases.

Building on the spectral learning results of previous chapters, Chapter 6 presents a spectral approach to the sentiment regression problem introduced in Chapter 5. A common approach to this type of problem is to identify a set of "sentiment" words, and then to apply a machine learning algorithm to learn to classify sentiment based on the presence/absence of those words. The spectral representations used in this dissertation, which are applicable to real functions of Boolean inputs, are a natural fit to this application. The spectral features (i.e., basis functions) extend the commonly used word presence model by allowing the learner to recognize how logical ANDs, ORs, and XORs of the word presence features affect overall sentiment. Building on the results of Chapters 3 and 4, Chapter 6 presents a new spectral learning algorithm for sentiment analysis and compares its performance to that of other learning algorithms. Chapter 6 also shows how the spectral learning algorithm can be used as a feature selector, and how other algorithms can improve their performance by using the spectral features as inputs.

Finally, based on the observation in Chapter 2 that MAX-2-SAT (the maximum satisfiability problem, with the constraint that there are two literals per clause) can be reduced to the problem of finding the largest coefficient in a spectral representation, Chapter

7 explores the potential benefit of applying algorithms and methods for solving MAX-SAT problems to the problem of finding large coefficients, and vice versa. It presents results of MAX-SAT-inspired coefficient search algorithms, and also presents results of applying a coefficient search algorithm to satisfiability problems.

(Portions of Chapter 2 appeared previously in the proceedings of the *2008 AAAI Workshop on Search in Artificial Intelligence and Robotics* [Drake and Ventura, 2008] and in the proceedings of the *2009 International Joint Conference on Artificial Intelligence* [Drake and Ventura, 2009]. Chapter 5 appeared previously in the proceedings of the *2008 IEEE International Conference on Semantic Computing* [Drake et al., 2008].)

Chapter 2

Search Techniques for Spectral Learning

Abstract

Spectral learning algorithms learn an unknown function by learning a spectral representation of the function. Typically, the algorithm will attempt to do this by identifying the large coefficients of the function's spectral representation. Unfortunately, the number of spectral coefficients is exponential in the number of input features, making standard approaches to spectral computation impractical for large learning problems. In this paper, we analyze the problem of finding large spectral coefficients and introduce practical search algorithms, both complete and incomplete, for finding large coefficients in real-world settings. Although we show that the coefficient search problem is NP-Complete for the class of spectral representations under consideration, the search algorithms we present perform well in practice.

2.1 Introduction

Spectral learning algorithms learn an unknown function by learning a spectral representation of the function. Spectral learning methods have been studied extensively in learning theory, where algorithms that learn Fourier representations have been used to prove many interesting learnability results [Blum et al., 1994, Bshouty and Tamon, 1996, Jackson, 1997, Jackson et al., 2002, Kushilevitz and Mansour, 1993, Klivans et al., 2004, Linial et al., 1993, Mansour, 1995]. More recently, variations of these algorithms have been effectively applied in real-world settings [Drake and Ventura, 2005, Kargupta and Park, 2001, 2004, Kargupta et al., 2000, 2002, Mansour and Sahar, 2000, Park et al., 2001].

In order to learn effectively, a spectral learning algorithm must be able to identify the large coefficients of a function's spectral representation. This can be problematic if a learning problem has many input features, as the number of spectral coefficients is exponential in the number of inputs, and computing the entire spectrum is feasible only for small problems. In some cases, knowledge of the class of functions being learned may allow the algorithm to restrict its attention to a small subset of the coefficients (e.g., to low-order coefficients) [Bshouty and Tamon, 1996, Jackson et al., 2002, Klivans et al., 2004, Kargupta and Park, 2001, 2004, Kargupta et al., 2000, 2002, Linial et al., 1993, Park et al., 2001]. In general, however, the large coefficients can be anywhere in the spectrum. Algorithms have been developed for this more general case [Blum et al., 1994, Jackson, 1997, Kushilevitz and Mansour, 1993, Mansour, 1995, Mansour and Sahar, 2000], but they either rely on assumptions that are not typical of real-world scenarios (e.g., assuming a uniform distribution over examples) or can only be applied in limited domains (e.g., where an oracle is present to provide examples at training time).

This paper considers the problem of finding large coefficients in the common real-world scenario in which a learner is asked to learn an unknown function from an arbitrary, fixed set of labeled examples (i.e., in an agnostic learning setting). In particular, it discusses

the hardness of the coefficient search problem and presents a practical approach to finding large spectral coefficients in real-world settings.

The remainder of this paper is organized as follows: Section 2.2 provides background information on spectral representations and spectral learning. Section 2.3 discusses the hardness of the coefficient search problem, showing that the problem of finding a large coefficient is NP-Complete for a class of spectral representations that includes the Fourier and other representations. Section 2.4 introduces a method for obtaining bounds on coefficient size in regions of a spectral representation, and Sections 2.5 and 2.6 show how this technique can be incorporated into practical algorithms for finding large coefficients. Section 2.7 shows that these algorithms perform well in practice, as they usually only need to explore a small fraction of the coefficient search space to find the largest coefficients. Finally, Section 2.8 analyzes some of the characteristics of learning problems that affect the performance of the algorithms.

2.2 Background

Before describing the spectral learning approach and the problem of finding large coefficients, some background on the spectral representations being considered here is required.

2.2.1 Spectral Representations

The space of functions considered in this paper is the set F of functions of the form $f : \{0,1\}^n \rightarrow \mathbb{R}$. A basis B for F is a linearly-independent subset of F that is capable of expressing any $f \in F$ through some linear combination. There will be 2^n functions $\phi_\alpha : \{0,1\}^n \rightarrow \mathbb{R}$ in such a basis, and we assume that each can be uniquely identified by its n-digit binary label α. Then, any f can be expressed as

$$f(x) = \sum_{\alpha \in \{0,1\}^n} \hat{f}(\alpha)\phi_\alpha(x) \tag{2.1}$$

where $\hat{f}$ gives the unique linear combination of the basis functions that is equivalent to f. Each $\hat{f}(\alpha)$ is a spectral coefficient that corresponds to basis function ϕ_α.

The coefficients of a spectral representation in basis B can be obtained by the following:

$$\hat{f}(\alpha) = \sum_{x \in \{0,1\}^n} f(x)\phi_x^{-1}(\alpha) \tag{2.2}$$

where $\phi_x^{-1} : \{0,1\}^n \to \mathbb{R}$ is the x^{th} basis function in B^{-1}, the inverse of basis B. For any basis B, B^{-1} is the unique basis for which the following holds:

$$\forall \alpha, \beta \left(\sum_{x \in \{0,1\}^n} \phi_x(\alpha)\phi_\beta^{-1}(x) = \sum_{x \in \{0,1\}^n} \phi_x^{-1}(\beta)\phi_\alpha(x) = \begin{cases} 1 : \text{if } \alpha = \beta \\ 0 : \text{if } \alpha \neq \beta \end{cases} \right)$$

(This is analogous to the fact that the inverse of a square matrix A is the matrix A^{-1} such that $AA^{-1} = A^{-1}A = I$, where I is the identity matrix. Note that ϕ_α^{-1} does not denote the function that is the inverse of ϕ_α; that is, ϕ_α^{-1} is not a function of the form $\phi_\alpha^{-1} : \mathbb{R} \to \{0,1\}^n$ such that $\forall x \ \phi_\alpha^{-1}(\phi_\alpha(x)) = x$. Rather, for any basis function ϕ_α in a basis B, ϕ_α^{-1} denotes the basis function in B^{-1} that has the same label α.)

One well-known basis is the Fourier basis. The Fourier basis functions are defined by the following:

$$\chi_\alpha(x) = (-1)^{\sum_i \alpha_i x_i} = \begin{cases} +1 : \text{if } \sum_i \alpha_i x_i \text{ is even} \\ -1 : \text{if } \sum_i \alpha_i x_i \text{ is odd} \end{cases} \tag{2.3}$$

where $\alpha, x \in \{0,1\}^n$. Each χ_α is an XOR function, returning -1 iff the XOR of a particular subset of the inputs is true. The subset is implicitly defined by α. Since $\alpha_i x_i = 0$ when $\alpha_i = 0$, and since $\alpha_i x_i = x_i$ when $\alpha_i = 1$, the output of χ_α depends only on those x_i for which $\alpha_i = 1$. The basis functions of the inverse Fourier basis are defined as follows:

$$\chi_x^{-1}(\alpha) = \frac{1}{2^n}(-1)^{\sum_i \alpha_i x_i} = \begin{cases} +\frac{1}{2^n} : \text{if } \sum_i \alpha_i x_i \text{ is even} \\ -\frac{1}{2^n} : \text{if } \sum_i \alpha_i x_i \text{ is odd} \end{cases} \tag{2.4}$$

Note that the Fourier basis is identical to its inverse, modulo a constant factor. (The Fourier basis functions can be defined to output $\pm 1/\sqrt{2^n}$, in which case the inverse basis functions also output $\pm 1/\sqrt{2^n}$, and the Fourier basis is identical to its inverse.) This is due to the fact that the Fourier basis is symmetric (i.e., $\chi_\alpha(x) = \chi_x(\alpha)$) and orthogonal (i.e., $\alpha \neq \beta \Rightarrow \sum_x \chi_\alpha(x)\chi_\beta(x) = 0$). Because of the symmetry and orthogonality, Fourier coefficients can be computed in any of the following ways:

$$\begin{aligned}\hat{f}(\alpha) &= \sum_{x\in\{0,1\}^n} f(x)\chi_x^{-1}(\alpha) = \frac{1}{2^n}\sum_{x\in\{0,1\}^n} f(x)\chi_x(\alpha) \\ &= \sum_{x\in\{0,1\}^n} f(x)\chi_\alpha^{-1}(x) = \frac{1}{2^n}\sum_{x\in\{0,1\}^n} f(x)\chi_\alpha(x)\end{aligned} \tag{2.5}$$

However, this does not hold in general.

For example, consider the OR basis, which is patterned after the Fourier basis but uses OR functions instead of XOR functions. The OR basis functions are defined as follows:

$$\zeta_\alpha(x) = 1 - 2I\left(\sum_i \alpha_i x_i > 0\right) = \begin{cases} +1 : \text{if } \sum_i \alpha_i x_i = 0 \\ -1 : \text{if } \sum_i \alpha_i x_i > 0 \end{cases} \tag{2.6}$$

($I()$ is an indicator function that outputs 1 if the enclosed expression is true and outputs 0 otherwise.) These functions compute the OR of those inputs for which $\alpha_i = 1$. The basis functions of the inverse OR basis are defined by the following:

$$\zeta_x^{-1}(\alpha) = \begin{cases} \frac{1}{2} & : \text{if } (x = 0^n \wedge \alpha = 0^n) \\ \frac{1}{2}(-1)^{n+\sum_i \alpha_i + x_i} & : \text{if } (x \neq 0^n \vee \alpha \neq 0^n) \wedge \forall i(x_i = 1 \vee \alpha_i = 1) \\ 0 & : \text{if } (x \neq 0^n \vee \alpha \neq 0^n) \wedge \exists i(x_i = 0 \wedge \alpha_i = 0) \end{cases} \tag{2.7}$$

The OR basis is not orthogonal, so it is not equivalent to its inverse. It is symmetric, however, so either of the following can be used to compute the coefficients of an OR representation:

$$\hat{f}(\alpha) = \sum_{x \in \{0,1\}^n} f(x)\zeta_x^{-1}(\alpha) = \sum_{x \in \{0,1\}^n} f(x)\zeta_\alpha^{-1}(x) \tag{2.8}$$

Finally, consider the AND basis, which is neither symmetric nor orthogonal. It is also patterned after the Fourier basis, and its basis functions compute the AND of those inputs for which $\alpha_i = 1$:

$$\xi_\alpha(x) = 1 - 2I\left(\sum_i \alpha_i x_i < \sum_i \alpha_i\right) = \begin{cases} +1 : \text{if } \sum_i \alpha_i x_i = \sum_i \alpha_i \\ -1 : \text{if } \sum_i \alpha_i x_i < \sum_i \alpha_i \end{cases} \tag{2.9}$$

(Note that the AND functions output -1 when the AND is false, while the OR and XOR functions output -1 when the OR or XOR is true. Defining the functions this way will be convenient for the algorithms presented later.) The basis functions of the inverse AND basis are defined as follows:

$$\xi_x^{-1}(\alpha) = \begin{cases} \frac{1}{2} & : \text{if } (x = 1^n \wedge \alpha = 0^n) \\ \frac{1}{2}(-1)^{\sum_i \alpha_i + x_i} & : \text{if } (x \neq 1^n \vee \alpha \neq 0^n) \wedge \forall i(x_i = 0 \vee \alpha_i = 1) \\ 0 & : \text{if } (x \neq 1^n \vee \alpha \neq 0^n) \wedge \exists i(x_i = 1 \wedge \alpha_i = 0) \end{cases} \tag{2.10}$$

Since the AND basis is neither symmetric nor orthogonal, the coefficients of an AND representation can only be computed by the following:

$$\hat{f}(\alpha) = \sum_{x \in \{0,1\}^n} f(x)\xi_x^{-1}(\alpha) \tag{2.11}$$

For any basis $B = \{\phi_\alpha : \alpha \in \{0,1\}^n\}$, the transpose of B is defined to be the basis $B^T = \{\phi_\alpha^T : \alpha \in \{0,1\}^n\}$ such that $\forall \alpha, x(\phi_\alpha(x) = \phi_x^T(\alpha))$. (This is analogous to the transpose of a matrix A being the matrix A^T where $\forall i, j(A_{ij} = A_{ji}^T)$.) We will use the notation B^{-T}

to denote the basis that is the inverse of the transpose of B (or, equivalently, the transpose of the inverse of B). Therefore, $B^{-T} = (B^T)^{-1} = (B^{-1})^T$.

Note that if B is symmetric then $B = B^T$. Therefore, the following relationships exist between the OR and XOR bases and their transposes:

$$\chi_\alpha^T(x) = \chi_\alpha(x) = \chi_x(\alpha) = \chi_x^T(\alpha)$$
$$\chi_\alpha^{-T}(x) = \chi_\alpha^{-1}(x) = \chi_x^{-1}(\alpha) = \chi_x^{-T}(\alpha)$$
$$\zeta_\alpha^T(x) = \zeta_\alpha(x) = \zeta_x(\alpha) = \zeta_x^T(\alpha)$$
$$\zeta_\alpha^{-T}(x) = \zeta_\alpha^{-1}(x) = \zeta_x^{-1}(\alpha) = \zeta_x^{-T}(\alpha)$$

For the AND basis and its transpose, on the other hand, only the following hold:

$$\xi_\alpha^T(x) = \xi_x(\alpha)$$
$$\xi_\alpha^{-T}(x) = \xi_x^{-1}(\alpha)$$

2.2.2 Spectral Learning

For the task of learning an unknown function f from a set X of $\langle x, f(x)\rangle$ examples, spectral learning algorithms typically attempt to learn f by approximating its spectral representation, $\hat{f}$, from X. For example, the coefficients of the AND, OR, and XOR bases can be approximated by the following:

$$\hat{f}_X(\alpha) = \frac{2^n}{|X|} \sum_{\langle x, f(x)\rangle \in X} f(x)\phi_x^{-1}(\alpha) \tag{2.12}$$

where $\hat{f}_X$ is the spectral representation of X, or the approximation of $\hat{f}$ from X. Equation 2.12 provides an increasingly accurate approximation as $|X|$ increases.

Since there are 2^n basis functions for an n-input function, it is impractical to use all basis functions unless n is small. Therefore, spectral learning algorithms will typically use

only those basis functions whose estimated coefficients are largest, as these basis functions contribute the most to the linear combination. If m basis functions are used, then the set A of the labels of those basis functions is given by:

$$A = \underset{\alpha_1,\ldots,\alpha_m}{\operatorname{argmax}} \sum_{\alpha \in \{\alpha_1,\ldots,\alpha_m\}} \left| \hat{f}_X(\alpha) \right| \tag{2.13}$$

and a spectral learner's approximation of f is given by:

$$f(x) \approx \sum_{\alpha \in A} \hat{f}_X(\alpha) \phi_\alpha(x) \tag{2.14}$$

Although this approach to spectral learning is the most common, it is not always the most effective. Instead of trying to approximate $\hat{f}$, some spectral learning approaches attempt to learn a spectral representation $\hat{g}$ such that $g \approx f$:

$$f(x) \approx g(x) = \sum_{\alpha} \hat{g}(\alpha) \phi_\alpha(x) \tag{2.15}$$

Often, this is a more effective approach, meaning that g is a better approximation of f, even though there may have been no explicit attempt to make $\hat{g}$ resemble $\hat{f}$.

For example, one alternative is to view spectral learning as a feature selection problem, where the basis functions are features and spectral learning is the process of selecting and combining the best features. With this perspective, one might try to select basis functions that are similar to f. Squared error is a computationally convenient similarity metric that leads to the following:

$$A = \underset{\alpha_1,\ldots,\alpha_m}{\operatorname{argmin}} \sum_{\alpha \in \{\alpha_1,\ldots,\alpha_m\}} \left(\min_{\hat{g}_X(\alpha)} \sum_{\langle x, f(x) \rangle \in X} \left(f(x) - \hat{g}_X(\alpha) \phi_\alpha(x) \right)^2 \right) \tag{2.16}$$

For each ϕ_α, the value of $\hat{g}_X(\alpha)$ that minimizes squared error over X is given by:

$$\hat{g}_X(\alpha) = \frac{1}{\|\phi_\alpha\|_X^2} \sum_{\langle x, f(x)\rangle \in X} f(x)\phi_\alpha(x) \tag{2.17}$$

where $\|\phi_\alpha\|_X^2 = \sum_{\langle x,f(x)\rangle \in X} (\phi_\alpha(x))^2$. (See Theorem 8 in the appendix for details.) If $\forall \alpha, x(\phi_\alpha(x) = \pm k)$ for some $k \in \mathbb{R}$, then Equation 2.17 simplifies to:

$$\hat{g}_X(\alpha) = \frac{1}{k^2|X|} \sum_{\langle x, f(x)\rangle \in X} f(x)\phi_\alpha(x) \tag{2.18}$$

If $k = 1$, as is the case for the AND, OR, and XOR bases, then Equation 2.17 simplifies further to:

$$\hat{g}_X(\alpha) = \frac{1}{|X|} \sum_{\langle x, f(x)\rangle \in X} f(x)\phi_\alpha(x) \tag{2.19}$$

For any k, if $\forall \alpha, x(\phi_\alpha(x) = \pm k)$, then $|\hat{g}_X(\alpha)|$ is inversely proportional to $\sum_X (f(x) - \hat{g}_X(\alpha)\phi_\alpha(x))^2$, and Equation 2.16 is equivalent to the following:

$$A = \underset{\alpha_1,\ldots,\alpha_m}{\operatorname{argmax}} \sum_{\alpha \in \{\alpha_1,\ldots,\alpha_m\}} |\hat{g}_X(\alpha)| \tag{2.20}$$

Thus, for bases like the AND, OR, and XOR bases, selecting the basis functions with smallest squared error over X is equivalent to selecting the basis functions with the largest coefficients in $\hat{g}_X$. After the basis functions are selected, the coefficients can be set to optimize any metric, such as minimum squared error of the entire model over the training data:

$$\underset{\hat{g}_X(\alpha_1),\ldots,\hat{g}_X(\alpha_m)}{\operatorname{argmin}} \sum_{\langle x, f(x)\rangle \in X} \left(f(x) - \sum_{\alpha \in \{\alpha_1,\ldots,\alpha_m\}} \hat{g}_X(\alpha)\phi_\alpha(x) \right)^2 \tag{2.21}$$

Selecting basis functions that are good approximations of f is also the strategy used in boosting approaches to spectral learning [Jackson, 1997, Jackson et al., 2002].

For the Fourier/XOR basis, the basis function selection methods implied by Equations 2.13 and 2.20 are equivalent. That is, the basis functions whose coefficients are largest in the XOR representation of f are also the XOR functions that best approximate f (in terms of squared error). In general, however, this is not true, and one must typically choose which approach to follow. The remainder of this paper will focus on the minimum-squared-error approach, as it has been shown to perform better in practice [Drake and Ventura, 2011a].

For any spectral learning approach, the algorithms can be naturally applied to Boolean classification problems by encoding the outputs of positive and negative examples as -1.0 and 1.0, respectively, and using the sign of the model's output to make classifications:

$$f(x) \approx \begin{cases} false & : \text{if } \sum_{\alpha \in A} \hat{f}_X(\alpha)\phi_\alpha(x) \geq 0 \\ true & : \text{if } \sum_{\alpha \in A} \hat{f}_X(\alpha)\phi_\alpha(x) < 0 \end{cases} \tag{2.22}$$

2.2.3 The MSE Spectrum

In Equations 2.17-2.19, $\hat{g}_X$ approximates a spectrum $\hat{g}$ in which each $\hat{g}(\alpha)$ minimizes the squared difference between ϕ_α and f. This spectrum will be referred to as the Minimum Squared Error (MSE) spectrum. Stated precisely, the MSE spectrum of a function f with respect to basis $B = \{\phi_\alpha : \alpha \in \{0,1\}^n\}$ is denoted by $\check{f}$ and is defined by:

$$\check{f}(\alpha) = \frac{1}{\|\phi_\alpha\|^2} \sum_{x \in \{0,1\}^n} f(x)\phi_\alpha(x) \tag{2.23}$$

where $\|\phi_\alpha\|^2 = \sum_{x \in \{0,1\}^n} (\phi_\alpha(x))^2$. The approximation of $\check{f}$ from a set X of examples, or the MSE spectrum of X, is denoted by $\check{f}_X$ and given by:

$$\check{f}_X(\alpha) = \frac{1}{\|\phi_\alpha\|_X^2} \sum_{\langle x, f(x) \rangle \in X} f(x)\phi_\alpha(x) \tag{2.24}$$

where $\|\phi_\alpha\|_X^2 = \sum_{\langle x, f(x)\rangle \in X} (\phi_\alpha(x))^2$. Again, these definitions simplify for the AND, OR, and XOR bases:

$$\check{f}(\alpha) = \frac{1}{2^n} \sum_{x \in \{0,1\}^n} f(x)\phi_\alpha(x) \tag{2.25}$$

$$\check{f}_X(\alpha) = \frac{1}{|X|} \sum_{\langle x, f(x)\rangle \in X} f(x)\phi_\alpha(x) \tag{2.26}$$

Remember that $\check{f}$ does not provide a representation of f in basis B (unless $\check{f} = \hat{f}$). In general, $\check{f}$ provides a representation of f in basis B^{-1}:

$$f(x) = \sum_{\alpha \in \{0,1\}^n} \|\phi_\alpha\|^2 \check{f}(\alpha)\phi_x^{-1}(\alpha)$$

(See Theorem 9 in the appendix for details.) However, being able to represent f in B^{-1} is not the motivation for the MSE spectrum. Rather, it is interesting because for some bases (including the AND, OR, and XOR bases) large coefficients in the MSE spectrum correspond to basis functions in B that are good approximations of f.

2.2.4 Finding Large Spectral Coefficients

Whether a spectral learning approach is based on approximating a function's representation in a particular basis or on identifying and combining useful features, the core of a spectral learning algorithm is usually the process of finding large coefficients in some spectrum. However, since there are an exponential number of coefficients, brute-force approaches to computing and identifying large coefficients (such as the Fast Fourier Transform algorithm) are infeasible for large learning problems. Consequently, more sophisticated algorithms are needed.

2.3 Finding Large Coefficients is Hard

As algorithms are developed for finding large spectral coefficients, it will be useful to know when polynomial-time algorithms may or may not exist. This section shows that for the spectra considered in this paper, no polynomial-time algorithms exist in the general case (i.e., for an arbitrary data set), unless P = NP.

Definitions and Notation

The proof of the hardness of the coefficient search problem is based on reduction from MAX-2-SAT. MAX-2-SAT is a restricted version of MAX-SAT, the maximum satisfiability problem, which is the problem of determining the maximum number of clauses in a conjunctive normal form (CNF) expression that can be simultaneously satisfied by any truth assignment. In MAX-2-SAT, each clause may contain only two literals. Both MAX-SAT and MAX-2-SAT are known to be NP-complete problems. The following definition formalizes the MAX-2-SAT problem, casting it as a set membership problem.

Definition 1 (MAX-2-SAT)**.** *Given a set U of binary variables, a set C of CNF clauses over U such that each clause $c \in C$ contains 2 literals, and a scalar m, MAX-2-SAT $= \{\langle U, C, m\rangle :$ there exists a truth assignment for the variables in U that simultaneously satisfies at least m clauses in $C\}$.*

Definition 2 formalizes the problem of finding large coefficients in $\check{f}$ in a similar way.

Definition 2 (LARGE-$\check{f}$-COEF)**.** *For a given basis $B = \{\phi_\alpha : \alpha \in \{0,1\}^n\}$, if X is a set of $\langle x, f(x)\rangle$ examples with $x \in \{0,1\}^n$ and $f(x) \in \mathbb{R}$, then LARGE-$\check{f}$-COEF $= \{\langle n, X, \theta\rangle :$ there exists $\alpha \in \{0,1\}^n$ such that $|\check{f}_X(\alpha)| \geq |\theta|\}$.*

The following definitions describe "satisfiability" functions that will be useful for demonstrating the hardness of LARGE-$\check{f}$-COEF.

Definition 3 (f_c). *If $c = (l_i \vee l_j)$ is a clause of two literals where $l_i \in \{u_i, \neg u_i\}$ and $l_j \in \{u_j, \neg u_j\}$ for some variables $u_i, u_j \in \{u_1, u_2, ..., u_n\}$, then $f_c : \{0,1\}^n \to \{0,1\}$ is the function defined by*

$$f_c(x) = \begin{cases} 1 : \textit{if } c \textit{ is satisfied when } u_i = x_i \textit{ and } u_j = x_j \\ 0 : \textit{otherwise} \end{cases}$$

Definition 4 (f_C). *If C is a set of clauses such that each $c \in C$ is of the form $c = (l_i \vee l_j)$ with $l_i \in \{u_i, \neg u_i\}$ and $l_j \in \{u_j, \neg u_j\}$ for some variables $u_i, u_j \in \{u_1, u_2, ..., u_n\}$, then $f_C : \{0,1\}^n \to \mathbb{N}$ is the function defined by*

$$f_C(x) = \sum_{c \in C} f_c(x)$$

Thus, f_c returns 1 or 0 depending on whether or not clause c is satisfied by truth assignment x, and f_C gives the number of clauses in a set C that are satisfied by truth assignment x.

The following lemma shows that the spectral representation of any f_C is related to the spectral representations of its constituent f_c functions.

Lemma 1. *For any f_C and basis $B = \{\phi_\alpha : \alpha \in \{0,1\}^n\}$,*

$$\hat{f}_C(\alpha) = \sum_{c \in C} \hat{f}_c(\alpha)$$

Proof.

$$\begin{aligned}\hat{f}_C(\alpha) &= \sum_{x\in\{0,1\}^n} f_C(x)\phi_x^{-1}(\alpha) \\ &= \sum_{x\in\{0,1\}^n} \left(\sum_{c\in C} f_c(x)\right)\phi_x^{-1}(\alpha) \\ &= \sum_{c\in C}\sum_{x\in\{0,1\}^n} f_c(x)\phi_x^{-1}(\alpha) \\ &= \sum_{c\in C} \hat{f}_c(\alpha)\end{aligned}$$

□

NP-Completeness Theorem for Coefficient Search

The following theorem implicitly defines a class of spectral representations for which finding large MSE coefficients is an NP-complete problem.

Theorem 1. *Suppose* $B = \{\phi_\alpha : \alpha \in \{0,1\}^n\}$ *is a basis for functions of the form* $f : \{0,1\}^n \to \mathbb{R}$ *such that* $\forall\alpha, x(\phi_\alpha(x) = \pm k)$ *for some* $k \in \mathbb{R}$*, and suppose (1) that every* f_c *function has at most a polynomial (in* n*) number of non-zero coefficients in its representation in basis* B^T*, and (2) that given* c *these non-zero coefficients can be identified and computed in time polynomial in* n*. Then for basis* B*, LARGE-*$\check{f}$*-COEF is NP-complete.*

Proof. For any basis, LARGE-$\check{f}$-COEF is in NP because given $\alpha \in \{0,1\}^n$ it is verifiable in polynomial time whether $\left|\check{f}_X(\alpha)\right| \geq |\theta|$. The remainder of the proof will show that MAX-2-SAT reduces to LARGE-$\check{f}$-COEF for basis B.

Let $\langle U, C, m\rangle$ be an instance of MAX-2-SAT, and let f_C be the satisfiability function for C. Since $f_C : \{0,1\}^n \to \mathbb{N}$ (and since $\mathbb{N} \subset \mathbb{R}$), f_C can be expressed in basis B^T as

$$f_C(x) = \sum_{\alpha\in\{0,1\}^n} \hat{f}_C(\alpha)\phi_\alpha^T(x)$$

where $\hat{f}_C(\alpha) = \sum_x f(x)\phi_x^{-T}(\alpha)$ (Equation 2.2). Since $\hat{f}_C(\alpha) = \sum_{c \in C} \hat{f}_c(\alpha)$ (Lemma 1), and since each $\hat{f}_c$ has a polynomial number of non-zero coefficients that can be identified and computed in time polynomial in n, $\hat{f}_C$ has a polynomial number of non-zero coefficients that can be identified in time polynomial in n and $|C|$.

Therefore, we can construct an instance of LARGE-$\check{f}$-COEF, $\langle n, X, \theta \rangle$, from $\langle U, C, m \rangle$ in time polynomial in $|U|$, $|C|$, and m as follows: let $n = |U|$, let $\theta = m$, and build X from C by adding $\langle x, k^2|X|\hat{f}_C(x) \rangle$ to X for each x such that $\hat{f}_C(x) \neq 0$. (The $|X|$ in $\langle x, k^2|X|\hat{f}_C(x) \rangle$ refers to the eventual size of X, which is $|\{x : \hat{f}_C(x) \neq 0\}|$. k is the value such that $\forall \alpha, x(\phi_\alpha(x) = \pm k)$.)

All that remains is to show that $\langle U, C, m \rangle \in$ MAX-2-SAT if and only if $\langle n, X, \theta \rangle \in$ LARGE-$\check{f}$-COEF. Since $f(x) = k^2|X|\hat{f}_C(x)$ by construction,

$$\begin{aligned}
\check{f}_X(\alpha) &= \frac{1}{\|\phi_\alpha\|_X^2} \sum_{\langle x, f(x) \rangle \in X} f(x)\phi_\alpha(x) \\
&= \frac{1}{\|\phi_\alpha\|_X^2} \sum_{\langle x, f(x) \rangle \in X} \left(k^2|X|\hat{f}_C(x) \right) \phi_\alpha(x)
\end{aligned}$$

And, because $\forall \alpha, x(\phi_\alpha(x) = \pm k)$, $\forall \alpha (\|\phi_\alpha\|_X^2 = \sum_X \phi_\alpha(x)^2 = \sum_X k^2 = k^2|X|)$. Therefore,

$$\begin{aligned}
\check{f}_X(\alpha) &= \frac{1}{k^2|X|} \sum_{\langle x, f(x) \rangle \in X} k^2|X|\hat{f}_C(x)\phi_\alpha(x) \\
&= \sum_{\langle x, f(x) \rangle \in X} \hat{f}_C(x)\phi_\alpha(x)
\end{aligned}$$

Finally, because $\langle x, f(x) \rangle \notin X \iff \hat{f}_C(x) = 0$,

$$\begin{aligned}
\sum_{\langle x, f(x) \rangle \in X} \hat{f}_C(x)\phi_\alpha(x) &= \sum_{x \in \{0,1\}^n} \hat{f}_C(x)\phi_\alpha(x) \\
&= \sum_{x \in \{0,1\}^n} \hat{f}_C(x)\phi_x^T(\alpha) = f_C(\alpha)
\end{aligned}$$

Thus, $\check{f}_X(\alpha) = f_C(\alpha)$, so $\check{f}_X(\alpha)$ gives the number of clauses in C that are satisfied by truth assignment α. Therefore, $\langle U, C, m\rangle \in$ MAX-2-SAT if and only if $\check{f}_X(\alpha) \geq m$ for some $\alpha \in \{0,1\}^n$, which, since $m = \theta$, is true if and only if $\langle n, X, \theta\rangle \in$ LARGE-$\check{f}$-COEF. □

A consequence of Theorem 1 is that you can show that finding a large MSE coefficient is an NP-complete problem for any basis B for which $\forall\alpha, x(\phi_\alpha(x) = \pm k)$ by showing that every f_c function has a polynomial-size representation $\hat{f}_c$ in B^T, and that the non-zero coefficients of $\hat{f}_c$ can be identified and computed in polynomial time. The following corollaries show this for each of the representations considered in this paper. (Note that both the $\hat{f}$ (Equation 2.2) and $\check{f}$ (Equation 2.23) spectra are being used here. Specifically, the sparseness of every $\hat{f}_c$, where $\hat{f}_c$ is the representation of f_c in basis B^T, is used to show the hardness of finding large coefficients in $\check{f}$, the MSE spectrum of an arbitrary f with respect to basis B.)

Corollary 1. *For the Fourier/XOR basis, LARGE-$\check{f}$-COEF is NP-complete.*

Proof. The output of each f_c depends on only two inputs. Let i and j be these inputs, and let k be any other input. Imagine a partitioning of the input space into subsets $S_1, S_2, ..., S_{2^{n-1}}$, each of size 2, such that each subset pairs an input x with the input x' that differs from x only on input k. Then the XOR^T representation of f_c can be written as

$$\hat{f}_c(\alpha) = \sum_{\langle x,x'\rangle \in \{S_1, S_2, ... S_{2^{n-1}}\}} f_c(x)\chi_x^{-T}(\alpha) + f_c(x')\chi_{x'}^{-T}(\alpha)$$

For each $\langle x, x' \rangle$ pair, $f_c(x) = f_c(x')$ because $x_i = x'_i$ and $x_j = x'_j$. But, for any α such that $\alpha_k = 1$, $\chi_x^{-T}(\alpha) = -\chi_{x'}^{-T}(\alpha)$:

$$\begin{aligned}
\chi_x^{-T}(\alpha) &= (-1)^{\sum_m x_m \alpha_m} \\
&= (-1)^{x_k \alpha_k + \sum_{m \neq k} x_m \alpha_m} \\
&= -(-1)^{x'_k \alpha_k + \sum_{m \neq k} x_m \alpha_m} \\
&= -(-1)^{\sum_m x'_m \alpha_m} \\
&= -\chi_{x'}^{-T}(\alpha)
\end{aligned}$$

Thus, if $\alpha_k = 1$ then $f_c(x)\chi_x^{-T}(\alpha) + f_c(x')\chi_{x'}^{-T}(\alpha) = f_c(x)\chi_x^{-T}(\alpha) + f_c(x)(-\chi_x^{-T}(\alpha)) = 0$. Since this is true for each $\langle x, x' \rangle$ pair, $(\alpha_k = 1) \Rightarrow (\hat{f}_c(\alpha) = 0)$. And, since this is true for any $k \notin \{i, j\}$, $\hat{f}_c(\alpha) = 0$ unless $\alpha_k = 0$ for all $k \notin \{i, j\}$. There are four coefficients satisfying $\alpha_k = 0$ for all $k \notin \{i, j\}$ (one for each possible assignment of 0 and 1 to α_i and α_j). Therefore, each f_c has at most four non-zero coefficients in its XOR^T representation, and they are the coefficients for which $(k \notin \{i, j\}) \Rightarrow (\alpha_k = 0)$.

Now consider the computation of those coefficients. Suppose the input space is partitioned into four subsets, each of size 2^{n-2}, based on the values of x_i and x_j. Then,

$$\begin{aligned}
\hat{f}_c(\alpha) = &\sum_{\{x : x_i = 0 \wedge x_j = 0\}} f_c(x)\chi_x^{-T}(\alpha) + \sum_{\{x : x_i = 0 \wedge x_j = 1\}} f_c(x)\chi_x^{-T}(\alpha) \\
&+ \sum_{\{x : x_i = 1 \wedge x_j = 0\}} f_c(x)\chi_x^{-T}(\alpha) + \sum_{\{x : x_i = 1 \wedge x_j = 1\}} f_c(x)\chi_x^{-T}(\alpha)
\end{aligned}$$

Since f_c depends only on x_i and x_j, f_c is constant in each partition. And, for α such that $\alpha_k = 0$ for all $k \notin \{i, j\}$, χ_x^{-T} is also constant in each partition because $\chi_x^{-T}(\alpha) = (-1)^{\sum_m \alpha_m x_m} = (-1)^{\alpha_i x_i + \alpha_j x_j}$. Let $f_c(x_i, x_j)$ and $\chi_{x_i, x_j}^{-T}(\alpha)$ denote the values of $f_c(x)$ and $\chi_x^{-T}(\alpha)$, respectively, in each partition. Then for any α such that $\alpha_k = 0$ for all $k \notin \{i, j\}$,

$\hat{f}_c(\alpha)$ can be computed in polynomial time by

$$\begin{aligned}\hat{f}_c(\alpha) &= (2^{n-2})f_c(0,0)\chi_{0,0}^{-T}(\alpha) + (2^{n-2})f_c(0,1)\chi_{0,1}^{-T}(\alpha) \\ &\quad + (2^{n-2})f_c(1,0)\chi_{1,0}^{-T}(\alpha) + (2^{n-2})f_c(1,1)\chi_{1,1}^{-T}(\alpha)\end{aligned}$$

□

Corollary 2. *For the OR basis, LARGE-$\check{f}$-COEF is NP-complete.*

Proof. The output of each f_c depends on only two inputs. Let i and j be these inputs, and let k be any other input. Imagine a partitioning of the input space into subsets $S_1, S_2, ..., S_{2^{n-1}}$, each of size 2, such that each subset pairs an input x with the input x' that differs from x only on input k. Then the OR^T representation of f_c can be written as

$$\hat{f}_c(\alpha) = \sum_{\langle x,x'\rangle \in \{S_1,S_2,...S_{2^{n-1}}\}} f_c(x)\zeta_x^{-T}(\alpha) + f_c(x')\zeta_{x'}^{-T}(\alpha)$$

For each $\langle x, x'\rangle$ pair, $f_c(x) = f_c(x')$ because $x_i = x'_i$ and $x_j = x'_j$. But, for any α such that $\alpha_k = 1$, $\zeta_x^{-T}(\alpha) = -\zeta_{x'}^{-T}(\alpha)$:

$$\begin{aligned}\zeta_x^{-T}(\alpha) &= \begin{cases} \frac{1}{2}(-1)^{n+\sum_m \alpha_m + x_m} & : \text{if } \forall m(\alpha_m = 1 \vee x_m = 1) \\ 0 & : \text{if } \exists m(\alpha_m = 0 \wedge x_m = 0) \end{cases} \\ &= \begin{cases} \frac{1}{2}(-1)^{n+\alpha_k + x_k + \sum_{m\neq k} \alpha_m + x_m} & : \text{if } \forall m(\alpha_m = 1 \vee x_m = 1) \\ 0 & : \text{if } \exists m(\alpha_m = 0 \wedge x_m = 0) \end{cases} \\ &= \begin{cases} -\frac{1}{2}(-1)^{n+\alpha_k + x'_k + \sum_{m\neq k} \alpha_m + x_m} & : \text{if } \forall m(\alpha_m = 1 \vee x_m = 1) \\ 0 & : \text{if } \exists m(\alpha_m = 0 \wedge x_m = 0) \end{cases} \\ &= \begin{cases} -\frac{1}{2}(-1)^{n+\sum_m \alpha_m + x'_m} & : \text{if } \forall m(\alpha_m = 1 \vee x_m = 1) \\ 0 & : \text{if } \exists m(\alpha_m = 0 \wedge x_m = 0) \end{cases} \\ &= -\zeta_{x'}^{-T}(\alpha)\end{aligned}$$

Thus, if $\alpha_k = 1$ then $f_c(x)\zeta_x^{-T}(\alpha) + f_c(x')\zeta_{x'}^{-T}(\alpha) = f_c(x)\zeta_x^{-T}(\alpha) + f_c(x)(-\zeta_x^{-T}(\alpha)) = 0$. Since this is true for each $\langle x, x' \rangle$ pair, $(\alpha_k = 1) \Rightarrow (\hat{f}_c(\alpha) = 0)$. And, since this is true for any $k \notin \{i, j\}$, $\hat{f}_c(\alpha) = 0$ unless $\alpha_k = 0$ for all $k \notin \{i, j\}$. There are four coefficients satisfying $\alpha_k = 0$ for all $k \notin \{i, j\}$ (one for each possible assignment of 0 and 1 to α_i and α_j). Therefore, each f_c has at most four non-zero coefficients in its OR^T representation, and they are the coefficients for which $(k \notin \{i, j\}) \Rightarrow (\alpha_k = 0)$.

Now consider the computation of those coefficients. For any α, $\zeta_x^{-T}(\alpha) = 0$ unless either $x = 0^n$ or $\forall m(x_m = 1 \vee \alpha_m = 1)$. Therefore,

$$\hat{f}_c(\alpha) \quad = \sum_{\{x:(x=0^n)\vee\forall m(x_m=1\vee\alpha_m=1)\}} f_c(x)\zeta_x^{-T}(\alpha)$$

For any α such that $(m \notin \{i, j\}) \Rightarrow (\alpha_m = 0)$, the expression $\forall m(x_m = 1 \vee \alpha_m = 1)$ is true only if $(m \notin \{i, j\}) \Rightarrow (x_m = 1)$. Therefore, every non-zero $\hat{f}_c(\alpha)$ can be computed in polynomial time by

$$\hat{f}_c(\alpha) \quad = \sum_{\{x:(x=0^n)\vee\forall m((m\notin\{i,j\})\Rightarrow(x_m=1))\}} f_c(x)\zeta_x^{-T}(\alpha)$$

which sums $f_c(x)\zeta_x^{-T}(\alpha)$ over five x values ($x = 0^n$ plus each x satisfying $(m \notin \{i, j\}) \Rightarrow (x_m = 1)$). □

Corollary 3. *For the AND basis, LARGE-$\check{f}$-COEF is NP-complete.*

Proof. The output of each f_c depends on only two inputs. Let i and j be these inputs, and let k be any other input. Imagine a partitioning of the input space into subsets $S_1, S_2, ..., S_{2^{n-1}}$, each of size 2, such that each subset pairs an input x with the input x' that differs from x only on input k. Then the AND^T representation of f_c can be written as

$$\hat{f}_c(\alpha) = \sum_{\langle x,x'\rangle\in\{S_1,S_2,...S_{2^{n-1}}\}} f_c(x)\xi_x^{-T}(\alpha) + f_c(x')\xi_{x'}^{-T}(\alpha)$$

For each $\langle x, x' \rangle$ pair, $f_c(x) = f_c(x')$ because $x_i = x'_i$ and $x_j = x'_j$. But, for any α such that $\alpha_k = 0$, $\xi_x^{-T}(\alpha) = -\xi_{x'}^{-T}(\alpha)$:

$$\begin{aligned}
\xi_x^{-T}(\alpha) &= \begin{cases} \frac{1}{2}(-1)^{\sum_m \alpha_m + x_m} & : \text{if } \forall m(\alpha_m = 0 \vee x_m = 1) \\ 0 & : \text{if } \exists m(\alpha_m = 1 \wedge x_m = 0) \end{cases} \\
&= \begin{cases} \frac{1}{2}(-1)^{\alpha_k + x_k + \sum_{m \neq k} \alpha_m + x_m} & : \text{if } \forall m(\alpha_m = 0 \vee x_m = 1) \\ 0 & : \text{if } \exists m(\alpha_m = 1 \wedge x_m = 0) \end{cases} \\
&= \begin{cases} -\frac{1}{2}(-1)^{\alpha_k + x'_k + \sum_{m \neq k} \alpha_m + x_m} & : \text{if } \forall m(\alpha_m = 0 \vee x_m = 1) \\ 0 & : \text{if } \exists m(\alpha_m = 1 \wedge x_m = 0) \end{cases} \\
&= \begin{cases} -\frac{1}{2}(-1)^{\sum_m \alpha_m + x'_m} & : \text{if } \forall m(\alpha_m = 0 \vee x_m = 1) \\ 0 & : \text{if } \exists m(\alpha_m = 1 \wedge x_m = 0) \end{cases} \\
&= -\xi_{x'}^{-T}(\alpha)
\end{aligned}$$

Thus, if $\alpha_k = 0$, then $f_c(x)\xi_x^{-T}(\alpha) + f_c(x')\xi_{x'}^{-T}(\alpha) = f_c(x)\xi_x^{-T}(\alpha) + f_c(x)(-\xi_x^{-T}(\alpha)) = 0$. Since this is true for each $\langle x, x' \rangle$ pair, $(\alpha_k = 0) \Rightarrow (\hat{f}_c(\alpha) = 0)$. And, since this is true for any $k \notin \{i, j\}$, $\hat{f}_c(\alpha) = 0$ unless $\alpha_k = 1$ for all $k \notin \{i, j\}$. There are four coefficients satisfying $\alpha_k = 1$ for all $k \notin \{i, j\}$ (one for each possible assignment of 0 and 1 to α_i and α_j). Therefore, each f_c has at most four non-zero coefficients in its AND^T representation, and they are the coefficients for which $(k \notin \{i, j\}) \Rightarrow (\alpha_k = 1)$.

Now consider the computation of those coefficients. For any α, $\xi_x^{-T}(\alpha) = 0$ unless either $x = 0^n$ or $\forall m(\alpha_m = 0 \vee x_m = 1)$. Therefore,

$$\hat{f}_c(\alpha) = \sum_{\{x:(x=0^n) \vee \forall m(\alpha_m = 0 \vee x_m = 1)\}} f_c(x)\xi_x^{-T}(\alpha)$$

For any α such that $(m \notin \{i, j\}) \Rightarrow (\alpha_m = 1)$, the expression $\forall m(\alpha_m = 0 \vee x_m = 1)$ is true only if $(m \notin \{i, j\}) \Rightarrow (x_m = 1)$. Therefore, every non-zero $\hat{f}_c(\alpha)$ can be computed in

polynomial time by

$$\hat{f}_c(\alpha) \quad = \quad \sum_{\{x:(x=0^n)\vee\forall m((m\notin\{i,j\})\Rightarrow(x_m=1))\}} f_c(x)\xi_x^{-T}(\alpha)$$

which sums $f_c(x)\xi_x^{-T}(\alpha)$ over five x values ($x = 0^n$ plus each x satisfying $(m \notin \{i,j\}) \Rightarrow (x_m = 1)$). □

2.4 Bounding Coefficient Size

Although the previous section suggests that polynomial-time algorithms for finding large coefficients do not exist in the general case, the following sections present search algorithms that perform well in practice. At the heart of the algorithms is a technique for bounding the size of the largest possible coefficient in a region of a spectral representation. With this technique, search algorithms can focus their efforts on promising areas of the search space.

2.4.1 Definitions & Notations

Before describing the coefficient bounding technique, some definitions and notation are needed.

Definition 5 (Spectral Regions (β))**.** *A partially-defined coefficient label $\beta \in \{0,1,*\}^n$ represents a region of a spectral representation. In particular, β represents the following set:* $\{\alpha : \alpha \in \{0,1\}^n \wedge \forall i((\alpha_i = \beta_i) \vee (\beta_i = *))\}$.

Each β represents the set of all coefficients labels that match β on its defined digits. For example, $\beta = *^n$ denotes the entire spectrum, while $\beta = 0*1*$ denotes the spectral region corresponding to the following set of coefficient labels: $\{0010, 0011, 0110, 0111\}$. The notation $\alpha \in \beta$ will be used to indicate an α in region β.

Definition 6 (β-Reduced Vectors ($r_\beta(\cdot)$))**.** *If $x \in \{0,1\}^n$ and $\beta \in \{0,1,*\}^n$, then $r_\beta(x) = x'$ where $x' \in \{0,1\}^{n-|\{i:\beta_i\neq *\}|}$ and $(\beta_i = *) \Rightarrow (x'_{i-\sum_{j=1}^{i} I(\beta_j\neq *)} = x_i)$.*

Informally, $r_\beta(x)$ reduces an n-dimensional vector x to an $(n-k)$-dimensional vector that is the result of removing each of k dimensions for which $\beta_i \neq *$. For example, if $x = 0010$ and $\beta = 1**0$, then $r_\beta(x) = r_{1**0}(0010) = 01$. If $x = 0010$ and $\beta = **1*$, then $r_\beta(x) = r_{**1*}(0010) = 000$.

Definition 7 (β-Reduced Data Sets (X_β))**.** *If X is a set of $\langle x, f(x)\rangle$ examples with $x \in \{0,1\}^n$ and $f(x) \in \mathbb{R}$, and $\beta \in \{0,1,*\}^n$, then a β-reduction of X is a set X_β of $\langle x, f(x)\rangle$ examples with $x \in \{0,1\}^{n-|\{i:\beta_i\neq *\}|}$ and $f(x) \in \mathbb{R}$ for which the following holds:*

$$(\alpha \in \beta) \Rightarrow \left(\|\phi_\alpha\|_X^2 \check{f}_X(\alpha) = \left\|\phi_{r_\beta(\alpha)}\right\|_{X_\beta}^2 \check{f}_{X_\beta}(r_\beta(\alpha))\right)$$

Note that

$$\begin{aligned}
&\left(\|\phi_\alpha\|_X^2 \check{f}_X(\alpha) = \left\|\phi_{r_\beta(\alpha)}\right\|_{X_\beta}^2 \check{f}_{X_\beta}(r_\beta(\alpha))\right) \\
&\qquad \Longleftrightarrow \left(\|\phi_\alpha\|_X^2 \left(\frac{1}{\|\phi_\alpha\|_X^2} \sum_{\langle x, f(x)\rangle \in X} f(x)\phi_\alpha(x)\right)\right. \\
&\qquad\qquad \left. = \left\|\phi_{r_\beta(\alpha)}\right\|_{X_\beta}^2 \left(\frac{1}{\left\|\phi_{r_\beta(\alpha)}\right\|_{X_\beta}^2} \sum_{\langle x', f(x')\rangle \in X_\beta} f(x')\phi_{r_\beta(\alpha)}(x')\right)\right) \\
&\qquad \Longleftrightarrow \left(\sum_{\langle x, f(x)\rangle \in X} f(x)\phi_\alpha(x) = \sum_{\langle x', f(x')\rangle \in X_\beta} f(x')\phi_{r_\beta(\alpha)}(x')\right)
\end{aligned}$$

Therefore, X_β is an $(n - |\{i : \beta_i \neq *\}|)$-dimensional data set whose un-normalized spectral representation is identical to the un-normalized spectrum of an n-dimensional data set X in region β.

2.4.2 Bounding Coefficient Size via β-Reductions

The coefficient bounding method uses β-reduced data sets to bound coefficient size in regions of a spectral representation. The technique is based on the following theorems and their corollaries.

Theorem 2. *Let X be a set of $\langle x, f(x)\rangle$ examples with $x \in \{0,1\}^n$ and $f(x) \in \mathbb{R}$. Then for any basis $B = \{\phi_\alpha : \alpha \in \{0,1\}^n\}$,*

$$\max_\alpha \left|\check{f}_X(\alpha)\right| \leq \frac{\max_{\alpha,x\in X} |\phi_\alpha(x)|}{\min_{\alpha,x\in X} |\phi_\alpha(x)|^2 |X|} \sum_{\langle x,f(x)\rangle\in X} |f(x)| \tag{2.27}$$

Proof.

$$\begin{aligned}
\max_\alpha \left|\check{f}_X(\alpha)\right| &= \max_\alpha \left| \frac{1}{\|\phi_\alpha\|_X^2} \sum_{\langle x,f(x)\rangle\in X} f(x)\phi_\alpha(x) \right| \\
&\leq \max_\alpha \frac{1}{\|\phi_\alpha\|_X^2} \sum_{\langle x,f(x)\rangle\in X} |f(x)|\,|\phi_\alpha(x)| \\
&\leq \left(\max_\alpha \frac{1}{\|\phi_\alpha\|_X^2} \right) \left(\max_\alpha \sum_{\langle x,f(x)\rangle\in X} |f(x)|\,|\phi_\alpha(x)| \right) \\
&\leq \left(\frac{1}{\min_{\alpha,x\in X} |\phi_\alpha(x)|^2 |X|} \right) \left(\max_\alpha \sum_{\langle x,f(x)\rangle\in X} |f(x)|\,|\phi_\alpha(x)| \right) \\
&\leq \left(\frac{1}{\min_{\alpha,x\in X} |\phi_\alpha(x)|^2 |X|} \right) \left(\max_{\alpha,x\in X} |\phi_\alpha(x)| \right) \sum_{\langle x,f(x)\rangle\in X} |f(x)| \\
&\leq \frac{\max_{\alpha,x\in X} |\phi_\alpha(x)|}{\min_{\alpha,x\in X} |\phi_\alpha(x)|^2 |X|} \sum_{\langle x,f(x)\rangle\in X} |f(x)|
\end{aligned}$$

□

Theorem 2 gives a bound on the size of the largest possible coefficient for a data set. For bases like the AND, OR, and XOR bases, for which $\max_{\alpha,x} |\phi_\alpha(x)| = \min_{\alpha,x} |\phi_\alpha(x)| = 1$, this bound simplifies.

Corollary 4. *Let X be a set of $\langle x, f(x)\rangle$ examples with $x \in \{0,1\}^n$ and $f(x) \in \mathbb{R}$ and let $\check{f}_X$ denote the MSE spectrum of X with respect to either the AND, OR, or XOR basis. Then*

$$\max_\alpha \left|\check{f}_X(\alpha)\right| \leq \frac{1}{|X|} \sum_{\langle x,f(x)\rangle\in X} |f(x)| \tag{2.28}$$

Theorem 2 applies to β-reduced data sets as well, and because of the relationship between the spectrum of X_β and the spectrum of a higher-dimensional data set X (see Definition 7), we get Theorem 3.

Theorem 3. *Let X be a set of $\langle x, f(x)\rangle$ examples with $x \in \{0,1\}^n$ and $f(x) \in \mathbb{R}$, let $\check{f}_X$ denote the MSE spectrum of X with respect to basis $B = \{\phi_\alpha : \alpha \in \{0,1\}^n\}$, and let $\beta \in \{0,1,*\}$ denote a region of $\check{f}_X$. Then*

$$\max_{\alpha\in\beta} \left|\check{f}_X(\alpha)\right| \leq \frac{\max_{\alpha\in\beta, x\in X_\beta} \left|\phi_{r_\beta(\alpha)}(x)\right|}{\min_{\alpha\in\beta, x\in X} \left|\phi_\alpha(x)\right|^2 |X|} \sum_{\langle x, f(x)\rangle \in X_\beta} |f(x)| \tag{2.29}$$

Proof. By Definition 7, for $\alpha \in \beta$, $\|\phi_\alpha\|_X^2 \check{f}_X(\alpha) = \left\|\phi_{r_\beta(\alpha)}\right\|_{X_\beta}^2 \check{f}_{X_\beta}(r_\beta(\alpha)$. Therefore,

$$\begin{aligned}
\max_{\alpha\in\beta} \left|\check{f}_X(\alpha)\right| &= \max_{\alpha\in\beta} \left| \frac{\left\|\phi_{r_\beta(\alpha)}\right\|_{X_\beta}^2}{\|\phi_\alpha\|_X^2} \check{f}_{X_\beta}(r_\beta(\alpha)) \right| \\
&= \max_{\alpha\in\beta} \left| \frac{\left\|\phi_{r_\beta(\alpha)}\right\|_{X_\beta}^2}{\|\phi_\alpha\|_X^2} \left(\frac{1}{\left\|\phi_{r_\beta(\alpha)}\right\|_{X_\beta}^2} \sum_{\langle x, f(x)\rangle \in X_\beta} f(x)\phi_{r_\beta(\alpha)}(x) \right) \right| \\
&= \max_{\alpha\in\beta} \left| \frac{1}{\|\phi_\alpha\|_X^2} \sum_{\langle x, f(x)\rangle \in X_\beta} f(x)\phi_{r_\beta(\alpha)}(x) \right| \\
&\leq \max_{\alpha\in\beta} \frac{1}{\|\phi_\alpha\|_X^2} \sum_{\langle x, f(x)\rangle \in X_\beta} |f(x)| \left|\phi_{r_\beta(\alpha)}(x)\right| \\
&\leq \left(\max_{\alpha\in\beta} \frac{1}{\|\phi_\alpha\|_X^2} \right) \left(\max_{\alpha\in\beta} \sum_{\langle x, f(x)\rangle \in X_\beta} |f(x)| \left|\phi_{r_\beta(\alpha)}(x)\right| \right) \\
&\leq \left(\frac{1}{\min_{\alpha\in\beta, x\in X} \left|\phi_\alpha(x)\right|^2 |X|} \right) \left(\max_{\alpha\in\beta} \sum_{\langle x, f(x)\rangle \in X_\beta} |f(x)| \left|\phi_{r_\beta(\alpha)}(x)\right| \right) \\
&\leq \left(\frac{1}{\min_{\alpha\in\beta, x\in X} \left|\phi_\alpha(x)\right|^2 |X|} \right) \left(\max_{\alpha\in\beta, x\in X_\beta} \left|\phi_{r_\beta(\alpha)}(x)\right| \right) \sum_{\langle x, f(x)\rangle \in X_\beta} |f(x)| \\
&\leq \frac{\max_{\alpha\in\beta, x\in X_\beta} \left|\phi_{r_\beta(\alpha)}(x)\right|}{\min_{\alpha\in\beta, x\in X} \left|\phi_\alpha(x)\right|^2 |X|} \sum_{\langle x, f(x)\rangle \in X_\beta} |f(x)|
\end{aligned}$$

□

Again, the bound given by Theorem 3 simplifies for the AND, OR, and XOR bases, because $\forall \alpha, x(|\phi_\alpha(x)| = 1)$.

Corollary 5. *Let X be a set of $\langle x, f(x)\rangle$ examples with $x \in \{0,1\}^n$ and $f(x) \in \mathbb{R}$, let $\check{f}_X$ denote the MSE spectrum of X with respect to either the AND, OR, or XOR basis, and let $\beta \in \{0,1,*\}$ denote a region of $\check{f}_X$. Then*

$$\max_{\alpha \in \beta} \left|\check{f}_X(\alpha)\right| \leq \frac{1}{|X|} \sum_{\langle x, f(x)\rangle \in X_\beta} |f(x)| \tag{2.30}$$

Theorem 3 and Corollary 5 suggest a method for obtaining coefficient bounds for any spectral region: reduce X to X_β and then compute the bound given by Equation 2.29 or 2.30. The following section shows how to efficiently obtain β-reduced data sets for the AND, OR, and XOR spectra.

2.4.3 Obtaining β-Reduced Data Sets

The derivation of β-reduction methods is based on the following lemma.

Lemma 2. *Let $\beta \in \{0,1,*\}^n$ denote a spectral region, let X be a set of $\langle x, f(x)\rangle$ examples with $x \in \{0,1\}^n$ and $f(x) \in \mathbb{R}$, let X' be a set of $\langle x', f(x')\rangle$ examples with $x' \in \{0,1\}^{n-|\{i:\beta_i \neq *\}|}$ and $f(x') \in \mathbb{R}$, and suppose there is a one-to-one mapping from X onto X' such that for each $\langle x, f(x)\rangle \in X$ there is a corresponding example $\langle x', f(x')\rangle \in X'$ such that $(\alpha \in \beta) \Rightarrow (f(x)\phi_\alpha(x) = f(x')\phi_{r_\beta(\alpha)}(x'))$. Then X' is a β-reduction of X.*

Proof. By definition,

$$\begin{aligned} \|\phi_\alpha\|_X^2 \, \check{f}_X(\alpha) &= \|\phi_\alpha\|_X^2 \left(\frac{1}{\|\phi_\alpha\|_X^2} \sum_{\langle x, f(x)\rangle \in X} f(x)\phi_\alpha(x) \right) \\ &= \sum_{\langle x, f(x)\rangle \in X} f(x)\phi_\alpha(x) \end{aligned}$$

Because of the one-to-one mapping from X onto X' such that $(\alpha \in \beta) \Rightarrow (f(x)\phi_\alpha(x) = f(x')\phi_{r_\beta(\alpha)}(x'))$, if $\alpha \in \beta$ then the sum over X can be replaced by the corresponding sum over X'. Therefore, for $\alpha \in \beta$,

$$\begin{aligned} \|\phi_\alpha\|_X^2 \, \check{f}_X(\alpha) &= \sum_{\langle x', f(x')\rangle \in X'} f(x')\phi_{r_\beta(\alpha)}(x') \\ &= \left(\frac{\left\|\phi_{r_\beta(\alpha)}\right\|_{X'}^2}{\left\|\phi_{r_\beta(\alpha)}\right\|_{X'}^2} \right) \sum_{\langle x', f(x')\rangle \in X'} f(x')\phi_{r_\beta(\alpha)}(x') \\ &= \left\|\phi_{r_\beta(\alpha)}\right\|_{X'}^2 \check{f}_{X'}(r_\beta(\alpha)) \end{aligned}$$

Thus, X' is a β-reduction of X (by Definition 7). □

Lemma 2 implies that a method for β-reducing a data set can be derived on an example-by-example basis. In other words, if you can show that for every $\langle x, f(x)\rangle$ and β there is an $\langle x', f(x')\rangle$ such that $(\alpha \in \beta) \Rightarrow (f(x)\phi_\alpha(x) = f(x')\phi_{r_\beta(\alpha)}(x'))$, then you have shown how any X can be β-reduced. Theorems 4-6 use this approach to implicitly define β-reduction methods for the AND, OR, and XOR bases.

The search algorithms presented later explore the space of coefficient labels in an iterative manner, setting the digits of β one-at-a-time. For computational efficiency, they also derive each β-reduced data set from a "parent" data set that has already had all but one of the dimensions for which $\beta_i \neq *$ removed by previous reductions. Consequently, Theorems 4-6 are presented with this one-dimension-at-a-time approach in mind, so they assume that β has a single defined digit (i.e., $|\{i : \beta_i \neq *\}| = 1$), or, equivalently, that all but one of the dimensions for which $\beta_i \neq *$ have previously been removed. This restriction does not affect generality, as any β-reduction involving k dimensions can be obtained by a sequence of k single-dimension reductions. (To see why, note that if $X_{\beta'}$ is a β'-reduction of X_β, which is a β-reduction of X, then $(\alpha \in \beta) \Rightarrow (\|\phi_\alpha\|_X^2 \check{f}_X(\alpha) = \|\phi_{r_\beta(\alpha)}\|_{X_\beta}^2 \check{f}_{X_\beta}(r_\beta(\alpha)) = \|\phi_{r_{\beta'}(\alpha)}\|_{X_{\beta'}}^2 \check{f}_{X_{\beta'}}(r_{\beta'}(\alpha)))$, so $X_{\beta'}$ is a β'-reduction of X.)

Theorem 4. *Let $\langle x, f(x)\rangle$ be an example with $x \in \{0,1\}^n$ and $f(x) \in \mathbb{R}$ and let $\beta \in \{0,1,*\}^n$ be a region of the MSE spectrum of the XOR basis such that $\beta_k \in \{0,1\}$ for some $k \in \{1,\ldots,n\}$, and $(i \in (\{1,\ldots,n\} \setminus k)) \Rightarrow (\beta_i = *)$. Then for all $\alpha \in \beta$:*

- $(\beta_k = 1 \wedge x_k = 1) \Rightarrow (f(x)\chi_\alpha(x) = (-f(x))\chi_{r_\beta(\alpha)}(r_\beta(x)))$
- $(\beta_k = 0 \vee x_k = 0) \Rightarrow (f(x)\chi_\alpha(x) = f(x)\chi_{r_\beta(\alpha)}(r_\beta(x)))$

Proof. Suppose $\beta_k = 1 \wedge x_k = 1$. Then $\alpha_k x_k = 1$ for all $\alpha \in \beta$. Therefore, for $\alpha \in \beta$,

$$\begin{aligned} f(x)\chi_\alpha(x) &= f(x)(-1)^{\sum_i \alpha_i x_i} \\ &= f(x)(-1)^{1+\sum_{i:i\neq k} \alpha_i x_i} \\ &= f(x)(-1)(-1)^{\sum_{i:i\neq k} \alpha_i x_i} \\ &= (-f(x))(-1)^{\sum_{i:i\neq k} \alpha_i x_i} \\ &= (-f(x))\chi_{r_\beta(\alpha)}(r_\beta(x)) \end{aligned}$$

Now suppose $\beta_k = 0 \vee x_k = 0$. Then for all $\alpha \in \beta$, $\alpha_k x_k = 0$. Therefore, for $\alpha \in \beta$,

$$\begin{aligned} f(x)\chi_\alpha(x) &= f(x)(-1)^{\sum_i \alpha_i x_i} \\ &= f(x)(-1)^{\sum_{i:i\neq k} \alpha_i x_i} \\ &= f(x)\chi_{r_\beta(\alpha)}(r_\beta(x)) \end{aligned}$$

□

Theorem 5. *Let $\langle x, f(x)\rangle$ be an example with $x \in \{0,1\}^n$ and $f(x) \in \mathbb{R}$ and let $\beta \in \{0,1,*\}^n$ be a region of the MSE spectrum of the OR basis such that $\beta_k \in \{0,1\}$ for some $k \in \{1,\ldots,n\}$, and $(i \in (\{1,\ldots,n\} \setminus k)) \Rightarrow (\beta_i = *)$. Then for all $\alpha \in \beta$:*

- $(\beta_k = 1 \wedge x_k = 1) \Rightarrow (f(x)\zeta_\alpha(x) = (-f(x))\zeta_{r_\beta(\alpha)}(0^{n-1}))$
- $(\beta_k = 0 \vee x_k = 0) \Rightarrow (f(x)\zeta_\alpha(x) = f(x)\zeta_{r_\beta(\alpha)}(r_\beta(x)))$

Proof. Suppose $\beta_k = 1 \wedge x_k = 1$. Then $\alpha_k x_k = 1$ for all $\alpha \in \beta$, so $\sum_i \alpha_i x_i > 0$ for all $\alpha \in \beta$. Therefore, for $\alpha \in \beta$,

$$\begin{aligned} f(x)\zeta_\alpha(x) &= f(x)\left(1 - 2I\left(\sum_i \alpha_i x_i > 0\right)\right) \\ &= f(x)\left(1 - 2(1)\right) \\ &= -f(x) \end{aligned}$$

And, since $\forall\alpha(\xi_{r_k(\alpha)}(0^{n-1}) = 1)$,

$$f(x)\zeta_\alpha(x) = (-f(x))\zeta_{r_\beta(\alpha)}(0^{n-1})$$

Now suppose $\beta_k = 0 \vee x_k = 0$. Then for all $\alpha \in \beta$, $\alpha_k x_k = 0$. Therefore, for $\alpha \in \beta$,

$$\begin{aligned} f(x)\zeta_\alpha(x) &= f(x)\left(1 - 2I\left(\sum_i \alpha_i x_i > 0\right)\right) \\ &= f(x)\left(1 - 2I\left(\sum_{i:i\neq k} \alpha_i x_i > 0\right)\right) \\ &= f(x)\zeta_{r_\beta(\alpha)}(r_\beta(x)) \end{aligned}$$

□

Theorem 6. *Let $\langle x, f(x)\rangle$ be an example with $x \in \{0,1\}^n$ and $f(x) \in \mathbb{R}$ and let $\beta \in \{0,1,*\}^n$ be a region of the MSE spectrum of the AND basis such that $\beta_k \in \{0,1\}$ for some $k \in \{1,\ldots,n\}$, and $(i \in (\{1,\ldots,n\} \setminus k)) \Rightarrow (\beta_i = *)$. Then for all $\alpha \in \beta$:*

- $(\beta_k = 1 \wedge x_k = 0) \Rightarrow (f(x)\xi_\alpha(x) = (-f(x))\xi_{r_\beta(\alpha)}(1^{n-1}))$
- $(\beta_k = 0 \vee x_k = 1) \Rightarrow (f(x)\xi_\alpha(x) = f(x)\xi_{r_\beta(\alpha)}(r_\beta(x)))$

Proof. Suppose $\beta_k = 1 \wedge x_k = 0$. Then $\alpha_k = 1$ but $\alpha_k x_k = 0$ for all $\alpha \in \beta$, so $\sum_i \alpha_i x_i < \sum_i \alpha_i$ for all $\alpha \in \beta$. Therefore, for $\alpha \in \beta$,

$$\begin{aligned} f(x)\xi_\alpha(x) &= f(x)\left(1 - 2I\left(\sum_i \alpha_i x_i < \sum_i \alpha_i\right)\right) \\ &= f(x)\left(1 - 2(1)\right) \\ &= -f(x) \end{aligned}$$

And, since $\forall\alpha(\xi_{r_\beta(\alpha)}(1^{n-1}) = 1)$,

$$f(x)\xi_\alpha(x) = (-f(x))\xi_{r_\beta(\alpha)}(1^{n-1})$$

Now suppose $\beta_k = 0 \vee x_k = 1$. Then for all $\alpha \in \beta$, $\alpha_k x_k = \alpha_k$. Therefore, for $\alpha \in \beta$,

$$\begin{aligned} f(x)\xi_\alpha(x) &= f(x)\left(1 - 2I\left(\sum_i \alpha_i x_i < \sum_i \alpha_i\right)\right) \\ &= f(x)\left(1 - 2I\left(\sum_{i:i\neq k} \alpha_i x_i < \sum_{i:i\neq k} \alpha_i\right)\right) \\ &= f(x)\xi_{r_\beta(\alpha)}(r_\beta(x)) \end{aligned}$$

□

Theorems 4, 5, and 6 imply that for each $\langle x, f(x)\rangle \in X$ an appropriate $\langle x', f(x')\rangle$ for X_β can be determined from the values of β_k and x_k, where k is the dimension being removed. For example, for the AND basis, if $\beta_k = 1$ and $x_k = 0$, then the appropriate example is $\langle 1^{n-1}, -f(x)\rangle$; otherwise, it is $\langle r_\beta(x), f(x)\rangle$.

Rules for determining which $\langle x', f(x')\rangle$ examples to add to X_β can be derived from knowledge of the class of functions that compose the basis and the generalizations that can be made about region β given x_k. For example, for the AND basis, if $\beta_k = 1$ and $x_k = 0$, then $\xi_\alpha(x) = -1$ for all $\alpha \in \beta$ because those basis functions compute an AND of inputs that includes x_k (since $\beta_k = 1$) and must be false (since $x_k = 0$). Therefore, for $\alpha \in \beta$,

$f(x)\xi_\alpha(x) = -f(x)$, and $\langle x', f(x')\rangle$ must satisfy $f(x')\xi_{r_\beta(\alpha)}(x') = -f(x)$ for all $\alpha \in \beta$. The example $\langle 1^{n-1}, -f(x)\rangle$ satisfies this condition, since the AND of any subset of 1^{n-1} will always be true, making $f(x')\xi_{r_\beta(\alpha)}(x') = (-f(x))(1) = -f(x)$ for all α.

Figure 2.1 presents a procedure for creating β-reduced data sets based on Theorems 4-6. Lines 2-7, 8-13, and 14-19 build β-reduced data sets for the XOR, OR, and AND bases, respectively.

Notice that the outputs of the examples added to X_β have the same magnitude as the original examples in X. Since the bound given by Equation 2.30 is based on the sum of the absolute values of the examples' outputs, these β-reduced sets are not immediately helpful for bounding coefficient size, as the absolute values of the outputs have not changed. However, they do become useful after applying an additional step based on the following theorem.

Theorem 7. *Suppose X contains a pair of examples $\langle x, f(x)\rangle$ and $\langle y, f(y)\rangle$ such that $x = y$, and suppose $X' = (X \setminus \{\langle x, f(x)\rangle, \langle y, f(y)\rangle\}) \cup \langle x, f(x) + f(y)\rangle$. Then for all α, $\|\phi_\alpha\|_X^2 \check{f}_X(\alpha) = \|\phi_\alpha\|_{X'}^2 \check{f}_{X'}(\alpha)$.*

Proof. By definition,

$$\begin{aligned}\|\phi_\alpha\|_X^2 \check{f}_X(\alpha) &= \|\phi_\alpha\|_X^2 \left(\frac{1}{\|\phi_\alpha\|_X^2} \sum_{\langle v, f(v)\rangle \in X} \phi_\alpha(v) f(v)\right) \\ &= \phi_\alpha(x)f(x) + \phi_\alpha(y)f(y) + \sum_{\langle v, f(v)\rangle \in X \setminus \{\langle x, f(x)\rangle, \langle y, f(y)\rangle\}} \phi_\alpha(v) f(v)\end{aligned}$$

Since $x = y$, $\phi_\alpha(x) = \phi_\alpha(y)$, so

$$\|\phi_\alpha\|_X^2 \check{f}_X(\alpha) = \phi_\alpha(x)(f(x) + f(y)) + \sum_{\langle v, f(v)\rangle \in X \setminus \{\langle x, f(x)\rangle, \langle y, f(y)\rangle\}} \phi_\alpha(v) f(v)$$

```
        β-Reduce(X, k, β_k, basis)
(1)     X_β ← ∅

(2)     if basis = XOR
(3)         for each ⟨x, f(x)⟩ ∈ X
(4)             if β_k = 1 ∧ x_k = 1
(5)                 X_β ← X_β ∪ ⟨r_β(x), −f(x)⟩
(6)             else
(7)                 X_β ← X_β ∪ ⟨r_β(x), f(x)⟩

(8)     if basis = OR
(9)         for each ⟨x, f(x)⟩ ∈ X
(10)            if β_k = 1 ∧ x_k = 1
(11)                X_β ← X_β ∪ ⟨0^{n−1}, −f(x)⟩
(12)            else
(13)                X_β ← X_β ∪ ⟨r_β(x), f(x)⟩

(14)    if basis = AND
(15)        for each ⟨x, f(x)⟩ ∈ X
(16)            if β_k = 1 ∧ x_k = 0
(17)                X_β ← X_β ∪ ⟨1^{n−1}, −f(x)⟩
(18)            else
(19)                X_β ← X_β ∪ ⟨r_β(x), f(x)⟩

(20)    for each ⟨x, f(x)⟩, ⟨x′, f(x′)⟩ ∈ X_β s.t. x = x′
(21)        X_β ← X_β \ {⟨x, f(x)⟩, ⟨x′, f(x′)⟩}
(22)        X_β ← X_β ∪ ⟨x, f(x) + f(x′)⟩

(23)    return X_β
```

Figure 2.1: Algorithm for β-reducing a data set X with respect to the MSE spectrum of the specified AND, OR, or XOR basis. By assumption, $\beta_k \in \{0, 1\}$ and $(i \neq k) \Rightarrow (\beta_i = *)$.

And, since $X' = (X \setminus \{\langle x, f(x)\rangle, \langle y, f(y)\rangle\}) \cup \langle x, f(x) + f(y)\rangle$,

$$\begin{aligned}
\|\phi_\alpha\|_X^2 \, \check{f}_X(\alpha) &= \sum_{\langle v, f(v)\rangle \in X'} \phi_\alpha(v) f(v) \\
&= \frac{\|\phi_\alpha\|_{X'}^2}{\|\phi_\alpha\|_{X'}^2} \sum_{\langle v, f(v)\rangle \in X'} \phi_\alpha(v) f(v) \\
&= \|\phi_\alpha\|_{X'}^2 \, \check{f}_{X'}(\alpha)
\end{aligned}$$

□

Informally, Theorem 7 says that any two examples with identical inputs can be "merged" (i.e., be replaced by a single example with the same input and an output that is the sum of original examples' outputs) without changing the un-normalized spectrum. Applying this result to β-reduced data sets leads to the following corollary.

Corollary 6. *If X_β is a β-reduction of X, and if X'_β is derived from X_β by replacing any $\langle x, f(x)\rangle$ and $\langle y, f(y)\rangle$ such that $x = y$ with $\langle x, f(x) + f(y)\rangle$, then X'_β is a β-reduction of X.*

Proof. If X_β is a β-reduction of X, then

$$(\alpha \in \beta) \Rightarrow \left(\|\phi_\alpha\|_X^2 \, \check{f}_X(\alpha) = \left\|\phi_{r_\beta(\alpha)}\right\|_{X_\beta}^2 \check{f}_{X_\beta}(r_\beta(\alpha)) \right)$$

And, by Theorem 7, $\left\|\phi_{r_\beta(\alpha)}\right\|_{X_\beta}^2 \check{f}_{X_\beta}(r_\beta(\alpha)) = \left\|\phi_{r_\beta(\alpha)}\right\|_{X'_\beta}^2 \check{f}_{X'_\beta}(r_\beta(\alpha))$ for all α. Therefore,

$$(\alpha \in \beta) \Rightarrow \left(\|\phi_\alpha\|_X^2 \, \check{f}_X(\alpha) = \left\|\phi_{r_\beta(\alpha)}\right\|_{X'_\beta}^2 \check{f}_{X'_\beta}(r_\beta(\alpha)) \right)$$

so X'_β is a β-reduction of X. □

The fact that examples can be merged in β-reduced sets is the key to the tightening of bounds on coefficient size. When a set X is reduced to X_β, several examples will typically end up with identical inputs. Often this will be because two examples differed only along dimensions that were removed. For the AND and OR bases, this will also frequently occur when examples' inputs are set to 1^{n-1} or 0^{n-1}. Whenever examples are merged, the coefficient bound for that region may shrink. Specifically, it will shrink whenever the signs of examples' outputs, $f(x)$ and $f(y)$, are different, because then $|f(x) + f(y)| < |f(x)| + |f(y)|$. During the β-reduction procedure, the signs of the examples' outputs tend to be flipped such that the outputs of examples that are merged have opposite signs when β represents a region containing only small coefficients and have matching signs when β contains a large coefficient.

In Figure 2.1, Lines 20-22 represent the example merging step. Note that in addition to providing the means to bound coefficient size, the merging of examples also speeds up computation, as the β-reduced data sets contain fewer examples as β becomes increasingly well-defined. This speedup is beneficial to the search algorithms presented in the following section.

Note also that the β-reduction procedure can be used to compute exact coefficient values. When applied to a 1-dimensional data set (i.e., when a data set has had all dimensions but one removed and the last undefined digit of β is being set), the procedure produces a 0-dimensional data set containing a single example, $\langle -, f(x) \rangle$, whose output is the value of the coefficient. No change is needed to the algorithm, except that there must be some way to represent a 0-dimensional input vector, which will be the same for each example. Like any β-reduced data set, the absolute value of the example provides a bound on coefficient size. In this case, however, the region consists of a single coefficient, and the bound is the exact size of the coefficient.

2.5 Coefficient Search Algorithms

Given the method for bounding coefficient size presented in the previous section, it is possible to design search algorithms that can find large coefficients without needing to compute the entire spectrum. This section describes two algorithms for finding large coefficients. The first is a complete branch-and-bound search algorithm that always finds the k largest coefficients, and the second is an incomplete beam search algorithm that is fast but may not find all of the largest coefficients.

The search space that both algorithms consider is the space of all fully- and partially-defined coefficient labels. This space can be viewed as a binary tree of labeled nodes in which the root node's label is $*^n$, the labels of children of internal nodes are the two labels that result from setting an undefined digit β_i of the parent's label to 0 or 1, respectively, and the leaf nodes' labels are completely defined. Thus, the root node represents the entire

```
      FindLargeCoefs-BranchAndBound(X, k)
(1)       A ← ∅
(2)       stack.push(CreateNode(∗^n, X))
(3)       while stack is not empty
(4)           node ← stack.pop()
(5)           if (|A| < k) ∨ (max_{α∈node.β} |f̌_X(α)| > min_{α∈A} |f̌_X(α)|)
(6)               if (node.β ∈ {0,1}^n)
(7)                   if (|A| = k)
(8)                       A ← A \ {argmin_{α∈A} |f̌_X(α)|}
(9)                   A ← A ∪ node.β
(10)              else
(11)                  i ← GetUndefinedDigit(node.β)
(12)                  stack.push(CreateChildNode(node, i, 1)}
(13)                  stack.push(CreateChildNode(node, i, 0)}
(14)      return A
```

Figure 2.2: The branch-and-bound search algorithm. It returns the labels of the k largest coefficients in the MSE spectrum of X.

spectrum, child nodes represent two halves of their parent's spectral region, and leaf nodes represent specific coefficients.

2.5.1 Branch-and-Bound Search Algorithm

The branch-and-bound search algorithm is shown in Figure 2.2. The algorithm uses a depth-first search to find the k large coefficients, and it uses the coefficient bounding method of the previous section to prune branches that cannot possibly lead to coefficients that are larger than the k largest coefficients found so far.

In Figure 2.2, a stack is used to implement the depth-first search, although recursion could be used as well. The algorithm builds and returns the set A of the labels of the k largest coefficients. At line 2, the root node is pushed onto the stack. Each node contains a label, β, and a corresponding β-reduced data set, X_β. For the root node, $\beta = *^n$ and $X_\beta = X$. At line 3, the algorithm enters a loop that continues until the stack is empty. In each iteration of the loop, the node on top of the stack is popped off (line 4). Then, Equation 2.30 is used to bound $\max_{\alpha \in node.\beta} |\check{f}_X(\alpha)|$ and determine whether any coefficients in region β could be larger than the k^{th}-largest coefficient found so far (line 5). If none could be larger,

then that node is ignored, pruning that branch of the search tree. (Note that no pruning can be done until k labels have been added to A.)

If there could be a coefficient in region β that is larger than the k^{th}-largest coefficient found so far, and if the node's label is completely defined (line 6), then the coefficient corresponding to that node's label is larger than the k^{th}-largest coefficient found so far, so that label is added to the solution (line 9). Before that, however, if A already contains k coefficients then the label of the smallest coefficient is removed from A (lines 7-8).

If, on the other hand, the node's label is not completely defined, then an undefined digit of the node's label is selected (line 11) and the child nodes that result from setting that digit to 0 or 1 are created and pushed on the stack (lines 12-13). CreateChildNode(*node*, i, v) returns the child of *node* in which β_i is set to v, with the child's X_β obtained from *node*'s data set by the procedure in Figure 2.1.

The algorithm in Figure 2.2 assumes coefficients of equal size are of equal value, and ties between coefficients are broken arbitrarily. However, when there are coefficients of equal size, one may wish to break ties in favor of low-order basis functions, where the order of a basis function is defined by: $order(\phi_\alpha) = \sum_i \alpha_i$. For the AND, OR, and XOR functions, the order of a function is the number of inputs to which the function applies its Boolean operator. For these functions, low-order functions are simpler, so favoring low-order functions biases a spectral learner towards simpler basis functions, which can improve performance [Drake and Ventura, 2011a]. This change can be implemented by modifying line 5 so that a branch is not pruned if a node's region may contain a coefficient of equal size but lower order than a coefficient whose label is in A, and by modifying line 8 so that higher-order functions are removed when there is a tie for smallest coefficient size.

Because the branch-and-bound algorithm only prunes a branch when it cannot possibly lead to a coefficient that is larger than the k largest coefficients found so far, the algorithm is complete, and it will always find the k largest coefficients. However, it may require $O(2^n)$ time to do so in the worst case, as there are $O(2^n)$ nodes in the search space. Consequently,

```
      FindLargeCoefs-BeamSearch(X, k, w)
(1)       currentNodes ← {CreateNode(∗^n, X)}
(2)       for j ← 1 to n
(3)           newNodes ← ∅
(4)           for each node ∈ currentNodes
(5)               i ← GetUndefinedDigit(node.β)
(6)               newNodes ← newNodes ∪ {CreateChildNode(node, i, 1)}
(7)               newNodes ← newNodes ∪ {CreateChildNode(node, i, 0)}
(8)           currentNodes ← ∅
(9)           while (|currentNodes| ≤ w ∧ |newNodes| > 0)
(10)              currentNodes ← currentNodes ∪ newNodes.removeBest()
(11)      A ← ∅
(12)      while (|A| ≤ k)
(13)          A ← A ∪ currentNodes.removeBest()
(14)      return A
```

Figure 2.3: The beam search algorithm. It returns the labels of the k largest coefficients in the MSE spectrum of X that are found by a beam search of width w.

the usefulness of the algorithm is based on the assumption that in practice large portions of the search space will be pruned. Results presented later suggest that this is a reasonable assumption.

(An alternative to the branch-and-bound algorithm that is also complete is a previously introduced best-first algorithm that explores the search space in an efficient, best-first manner [Drake and Ventura, 2005]. However, the best-first algorithm's worst-case memory complexity is $O(2^n|X|)$, so it can exhaust memory resources if the search gets too big. In contrast, the branch-and-bound algorithm's worst-case memory complexity is $O(n|X|)$, and its run time is comparable to that of the best-first algorithm. See [Drake and Ventura, 2009] for details.)

2.5.2 Beam Search

The beam search algorithm, shown in Figure 2.3, provides a greedy alternative to the complete branch-and-bound algorithm. It explores the search tree in a breadth-first manner, but at each level of the tree it prunes all but the best k nodes, ensuring that the number of nodes under consideration stays tractable.

As shown in Figure 2.3, the beam search algorithm maintains a set of current nodes that initially contains only the root node of the search tree (line 1). The algorithm iterates over each level of the tree (lines 2-10), and in each iteration, the children of every current node are generated (lines 3-7) and the set of current nodes becomes the best w children of the previous set of current nodes (lines 8-10). As in the branch-and-bound algorithm, GetUndefinedDigit(β) returns the index of an undefined digit in β, and CreateChildNode($node$, i, v) returns the child of $node$ in which β_i is set to v, with the child's β-reduced data set obtained from $node$'s data set by the procedure in Figure 2.1. For the $currentNodes$ and $newNodes$ sets, removeBest() removes and returns the node with the largest coefficient bound (computed by Equation 2.30). (As suggested previously, however, ties can be broken in favor of lower-order functions.) After n iterations, the current nodes will be leaf nodes, and the labels of the best k nodes are returned (lines 11-14).

The beam search algorithm may need to prune branches from the search tree before it is possible to know which regions contain the k largest coefficients. Consequently, the algorithm is incomplete, and it may not find all of the k largest coefficients. However, the algorithm will never visit more than $O(nw)$ nodes, where n is the number of input features and w is the beam width. In contrast, the branch-and-bound algorithm visits an indeterminate number of nodes in general, and it could visit $O(2^n)$ nodes in the worst case. This makes the beam search algorithm's run time much more predictable, and it allows it to be applied to larger problems, since the beam width can be reduced until the run time is acceptable. Of course, reducing beam size can affect the quality of the solution. Fortunately, results presented later suggest that it is possible to find good solutions even when the beam width is quite narrow.

2.6 Variable-Ordering Heuristic

In both the branch-and-bound and beam search algorithms, when a node is to be replaced by its children, an input i for which $\beta_i = *$ is selected to be set to 0 and 1 in the child nodes.

In Figures 2.2 and 2.3 this step is represented by the GetUndefinedDigit(β) function. The choice of i does not affect the correctness of either algorithm, so inputs could be processed in an arbitrary order. However, both algorithms perform better when an effective variable ordering heuristic is used.

For the branch-and-bound algorithm, a good variable ordering can dramatically reduce the number of nodes that are visited, as some variable orderings allow much more of the search space to be pruned. For the beam search algorithm, on the other hand, the variable ordering does not affect the number of nodes visited, but a good ordering can greatly improve the quality of the solution, as a good ordering will lead the beam search to regions of the spectrum that contain large coefficients.

In the experiments presented in the following section, both search algorithms choose an undefined digit by applying the following heuristic, in which $\beta_{i\leftarrow 0}$ and $\beta_{i\leftarrow 1}$ denote the labels that result from setting β_i to 0 and 1, respectively:

$$i \leftarrow \underset{i}{\operatorname{argmin}} \left(\max_{\alpha \in \beta_{i\leftarrow 0}} \left| \check{f}_X(\alpha) \right| + \max_{\alpha \in \beta_{i\leftarrow 1}} \left| \check{f}_X(\alpha) \right| \right) \tag{2.31}$$

This heuristic chooses the input that results in the tightest/smallest coefficient bounds (Equation 2.30) in the child regions. For the branch-and-bound search algorithm, obtaining tighter bounds more quickly makes it possible to prune branches higher up in the tree. For the beam search algorithm, obtaining tighter bounds more quickly makes it easier to determine which regions contain large coefficients. In contrast, consider an ordering that tightens the bounds as little as possible at each step. If bounds do not tighten quickly, then the branch-and-bound algorithm will have to expand more nodes before it can determine that a given branch can be pruned. An ordering that does not tighten bounds quickly also increases the chance that the beam search algorithm will prune regions containing good coefficients, since pruning decisions will have to be made when there is little information about the maximum coefficient size in each region.

2.7 Empirical Results

This section analyzes the performance of the search algorithms on several real-world data sets [Newman et al., 1998], which are summarized in Table 2.1. Each data set represents a Boolean classification problem. Where necessary, non-Boolean input features were converted into Boolean features. Each nominal input feature was converted into a group of m Boolean features (one for each possible value of the feature), where only the Boolean feature corresponding to the correct nominal value was true in an example. Each numeric input feature was converted into a single Boolean feature that indicated whether the value was above or below a threshold t. The set T of candidate thresholds for an input i was obtained by first sorting the set $V = \{v : \langle x, f(x)\rangle \in X \wedge x_i = v\}$ of observed x_i values into an ordered list $v_1, v_2, \ldots, v_{|V|}$ where $(i < j) \Rightarrow (v_i < v_j)$, and then using the midpoints between adjacent values in the list as candidate thresholds; in other words, $T = \{(v_i + v_{i+1})/2 : 1 \leq i < |V|\}$.The chosen threshold was the value that minimized the number of unavoidable misclassifications (assuming that classifications would be made based on that input alone). Stated precisely, t was chosen by

$$\operatorname*{argmin}_{t \in T} \left(\min\left(\left|X^1_{>t}\right|, \left|X^{-1}_{>t}\right|\right) + \min\left(\left|X^1_{\leq t}\right|, \left|X^{-1}_{\leq t}\right|\right)\right)$$

where $X^{f(x)}_{>t}$ and $X^{f(x)}_{\leq t}$ are the examples with output $f(x)$ for which $x_i > t$ and $x_i \leq t$, respectively. The number of inputs listed in Table 2.1 is the number of inputs after converting non-Boolean inputs to Boolean.

Tables 2.2 shows the average number of nodes expanded by the branch-and-bound algorithm, both when using the variable-ordering heuristic (B&B+H) and when using a random ordering (B&B), when finding the single largest coefficient and the 1,000 largest coefficients of the XOR spectrum. (A node expansion is when a node is replaced by its child nodes.) The total number of nodes that could have been expanded is also shown. The results suggest that in practice the branch-and-bound algorithm will only need to expand a small fraction of the exponentially-large search space. The benefit of the variable-ordering

Table 2.1: Data set summary.

Data Set	Inputs	Examples
Pima	8	768
WBC1	9	699
Voting	16	435
Heart	20	270
SPECT	22	267
German	24	1,000
WBC3	30	569
WBC2	32	198
Chess	38	3,196

Table 2.2: Average number of nodes expanded while searching for the largest coefficient and the largest 1,000 coefficients of the XOR spectrum, and the maximum number of nodes that could have been expanded. The branch-and-bound algorithm usually expands only a small fraction of the search space, especially when using the variable ordering heuristic (B&B+H).

Data Set	Max Nodes	1,000 Coefficients		1 Coefficient	
		B&B	B&B+H	B&B	B&B+H
Pima	255	n/a	n/a	21	25
WBC1	511	n/a	n/a	17	10
Voting	65,535	10,483	3,696	35	16
Heart	1,048,575	45,142	7,208	2,648	138
SPECT	4,194,303	278,474	161,529	6,093	878
German	16,777,215	42,403	3,129	2,762	108
WBC3	1,073,741,823	95,466	4,448	4,781	66
WBC2	4,294,967,295	388,768	3,307	214,774	347
Chess	274,877,906,943	901,647	35,268	319,666	1,334

heuristic can also be seen, as the number of expanded nodes often decreases dramatically when using the heuristic, especially on the larger problems.

Table 2.3 shows the average run time of the branch-and-bound algorithm, again both with and without the variable-ordering heuristic. For comparison, the time required to compute the spectrum with the Fast Walsh Transform algorithm (FWT) is also shown. (The FWT is a Boolean-input version of the general discrete Fast Fourier Transform algorithm.) The FWT is fast when the number of input dimensions is small, but it becomes impractical as the dimensionality increases. (Note that the run times marked with * are estimates, as the

Table 2.3: Average time required by the Fast Walsh Transform algorithm (FWT) and the branch-and-bound algorithm, both with (B&B+H) and without (B&B) the variable-ordering heuristic, to find the largest coefficient and the largest 1,000 coefficients of the XOR spectrum. The FWT is impractical for large problems, while the B&B algorithm's run time is significantly reduced by using the variable-ordering heuristic.

Data Set	1,000 Coefficients			1 Coefficient		
	FWT	B&B	B&B+H	FWT	B&B	B&B+H
Pima	n/a	n/a	n/a	0.0 s	0.0 s	0.0 s
WBC1	n/a	n/a	n/a	0.0 s	0.0 s	0.0 s
Voting	0.1 s	0.1 s	0.0 s	0.0 s	0.0 s	0.0 s
Heart	0.9 s	0.7 s	0.1 s	0.7 s	0.2 s	0.0 s
SPECT	4.1 s	5.6 s	4.6 s	4.0 s	0.5 s	0.1 s
German	18.6 s	3.0 s	0.1 s	14.8 s	0.5 s	0.0 s
WBC3	19.8 m*	8.1 s	0.1 s	15.8 m*	0.8 s	0.0 s
WBC2	1.3 h*	16.8 s	0.0 s	1.1 h*	10.5 s	0.0 s
Chess	84.7 h*	4.8 m	0.9 s	67.4 h*	2.1 m	0.4 s

standard FWT requires too much memory to be applied to those problems. The estimates were obtained by extrapolating from observed run times under the conservative assumption that run time would double each time n increased by 1.) The branch-and-bound algorithm, on the other hand, can find the largest coefficient quickly even when applied to the larger problems. The results also show that the variable-ordering heuristic results in smaller run times, suggesting that the benefit of expanding fewer nodes offsets the extra computation required to compute the heuristic at each node.

Since the beam search can be made very fast simply by narrowing the beam width, the quality of the solutions returned by the beam search algorithm must be evaluated. Table 2.4 shows the result of using the beam search algorithm to find one large coefficient and then using the corresponding basis function to make classifications on test data. The accuracies shown are the average test accuracies (over 100 trials) when training and testing on random 90%/10% training/test splits of the data. A bold highlight indicates a result that is not significantly worse than the result obtained with a complete search, as measured by a paired random permutation test ($p = 0.05$). As the table shows, the beam does not usually need to

Table 2.4: Learning accuracy when using a single basis function obtained by a beam search of the given width. A bold highlight indicates a result that is not significantly worse (statistically) than the infinite-beam result. In most cases, a relatively small beam is sufficient to match the accuracy obtained with an infinitely large beam.

	Beam Width				
Data Set	1	2	4	8	∞
Pima	67.2%	73.1%	**73.6%**	**73.8%**	**73.7%**
WBC1	83.7%	89.7%	**91.7%**	**91.2%**	**91.3%**
Voting	82.8%	**96.0%**	**96.0%**	**96.0%**	**96.0%**
Heart	53.3%	59.8%	**73.7%**	**74.7%**	**72.9%**
SPECT	59.5%	61.8%	75.0%	**77.6%**	**77.6%**
German	58.5%	68.8%	71.1%	71.9%	**72.7%**
WBC3	80.8%	**88.9%**	**88.9%**	**89.0%**	**87.6%**
WBC2	55.4%	54.4%	60.2%	61.0%	**72.2%**
Chess	62.8%	73.7%	74.7%	**75.0%**	**75.1%**

be very wide before the learning accuracy is about as good as the accuracy obtained when using a complete search.

In fact, sometimes the classification accuracy is higher when using the beam search. Although we expect that basis functions with larger coefficients will usually be better models of the unknown function f, a larger coefficient only implies greater correlation with X, and not necessarily with f. This uncertainty in the true sizes of coefficients is advantageous to the beam search, as its solutions, which may be sub-optimal with respect to X, may often be as good for the learning task as the solutions returned by a complete search.

2.8 Analysis

There are many factors that influence the performance of the branch-and-bound and beam search algorithms. This section highlights some of these factors, focusing on how they affect the number of nodes expanded by the branch-and-bound search algorithm.

2.8.1 Example Distribution

For every learning problem there is a probability distribution, usually unknown, that governs the likelihood that each example will be encountered. In this section, we assume that this distribution D is represented by a probability mass function $p_D : \{0,1\}^n \rightarrow [0,1]$ over the input space, where $p_D(x)$ is the probability that x would be the input of a randomly selected example. We also assume that the examples in a data set are i.i.d. (i.e., that they are selected independently at random according to p_D).

The distribution over the example space is one important factor in the performance of the search algorithms. In particular, the extent to which an example distribution deviates from a uniform distribution is important. We shall refer to this deviation as skew and define the skew of an example distribution D to be the average absolute difference between p_D and a uniform distribution:

$$skew(D) = \frac{1}{2^n} \sum_{x \in \{0,1\}^n} \left| p_D(x) - \frac{1}{2^n} \right| \tag{2.32}$$

In a uniform distribution U, $\forall x(p_U(x) = \frac{1}{2^n})$. In addition, $\forall i(\Pr_U(x_i = 1) = \frac{1}{2})$. Based on this fact, we shall define the skew of an input x_i to be the absolute difference between $\Pr_D(x_i = 1)$ and $\frac{1}{2}$:

$$skew_D(x_i) = \left| \Pr_D(x_i = 1) - \frac{1}{2} \right| \tag{2.33}$$

And, we define the average input skew of a distribution to be the average skew of each input:

$$ave\_input\_skew(D) = \frac{1}{n} \sum_{i=1}^{n} skew_D(x_i) \tag{2.34}$$

Figure 2.4 demonstrates how input skew can affect the number of nodes that are expanded during a search for the largest coefficient of a randomly selected 10-input function. In the figure, the average number of node expansions is plotted as a function of the skew of a single input, x_1, when the skew of all other inputs is 0. (Also shown are dashed

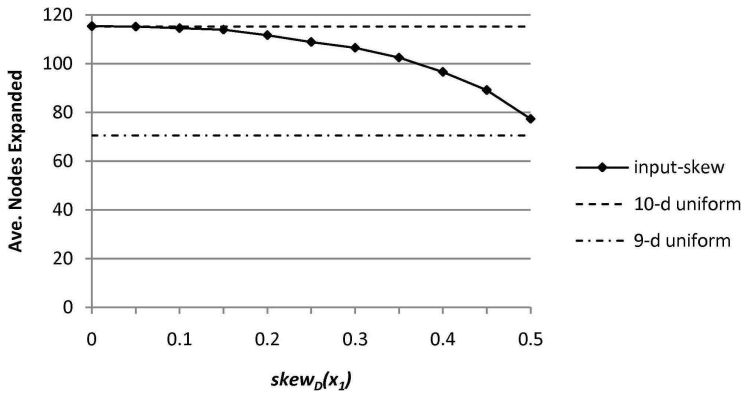


Figure 2.4: The average number of nodes expanded to find the largest XOR coefficient of a 10-dimensional learning problem as a function of the skew of a single input (with other inputs having 0 skew). As $skew_D(x_1)$ increases, the average number of expanded nodes approaches the average for a 9-dimensional problem.

lines indicating the average node expansions for 10- and 9-dimensional uniform distribution problems.) For each skew tested, the reported number of node expansions is an average over 1,000 data sets that were generated by randomly selecting a 10-input Boolean function $f : \{0,1\}^{10} \Rightarrow \{1,-1\}$, and then selecting 256 examples of the function at random according to the probability distribution $p_D(x) = \prod_i \Pr_D(x_i) = \Pr_D(x_1)(\frac{1}{2})^9$, where $\Pr_D(x_1 = 1) = \frac{1}{2} + skew_D(x_1)$. (Note that $p_D(x) = \prod_i \Pr_D(x_i)$ implies independence between input feature values. Note also that since these are Boolean input features, $\forall i(\Pr_D(x_i = 0) = 1 - \Pr_D(x_i = 1)$.) As the graph shows, the algorithm performs worst when the distribution is uniform, and the average number of expanded nodes decreases as $skew_D(x_1)$ increases from 0 to 0.5 (or, equivalently, as $\Pr_D(x_i = 1)$ increases from $\frac{1}{2}$ to 1).

The reduction in nodes expanded is related to the coefficient bounding method and variable ordering heuristic of the search algorithm. As described previously, the coefficient bounding method is based on merging examples whose input values are identical on inputs for which β is undefined. As $\Pr_D(x_1 = 1)$ becomes increasingly skewed towards 0 or 1, more examples tend to have the same value for x_1. Consequently, examples are more likely to merge earlier in the search, which means that coefficient bounds are more likely to tighten sooner as well. On the other hand, since more examples have the same value for x_1, splitting

the search space along dimension 1 does less in terms of tightening coefficient bounds, as removing input 1 is less likely to increase the number of examples that can merge. In the extreme case, where $\Pr_D(x_1 = 1)$ is 0 or 1, splitting along input 1 does not cause any examples to merge that would not already have been merged, and the coefficient bounds for the child regions of β are the same as for β. The search algorithm's variable ordering heuristic splits the search space along dimensions that tighten coefficient bounds the most, which implicitly causes the algorithm to avoid splitting the space along highly skewed dimensions. By saving skewed dimensions until the end, the algorithm is able to obtain tight coefficient bounds earlier in the search, allowing more of the search space to be pruned.

Note that node count decreases from the average number of node expansions for a 10-input uniform distribution problem to (almost) the number of expansions for a 9-input uniform distribution problem. When $\Pr_D(x_i = 1) = 1$, input i is completely irrelevant. Expanding a node along that dimension never results in reduced coefficient bounds, and the search algorithm's variable ordering heuristic causes input i to be the last input considered. The node count is only higher than the average for a 9-input problem because the algorithm must pass through the level of the tree that corresponds to splitting along dimension i before reaching the leaf nodes.

Figure 2.5 plots average node expansions as a function of average input skew (Equation 2.34). Here, as before, there is assumed to be independence between input feature values, so $p_D(x) = \prod_i \Pr_D(x_i)$. And, as before, results are averaged over 1,000 data sets that were generated by randomly selecting a function and then selecting 256 examples of the function at random according to $p_D(x)$. This time, however, all inputs could be skewed, and the plot shows results in terms of average skew over all inputs. As the plot shows, average node count initially decreases dramatically as average skew increases, until the average skew is around 0.25, after which node count increases slightly until the skew approaches the extreme case of total skew, where all examples have the same value for each input.

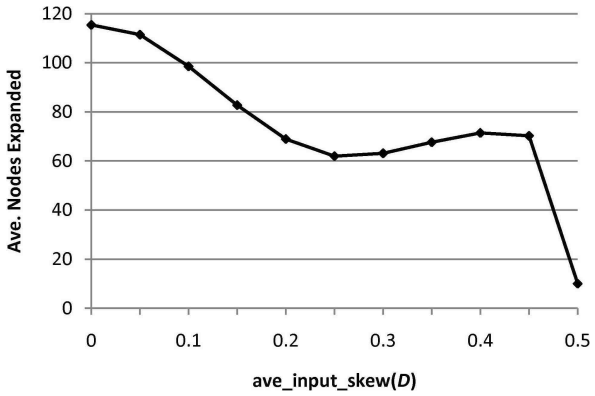


Figure 2.5: The average number of nodes expanded to find the largest XOR coefficient of a 10-dimensional learning problem as a function of the average input skew of the example distribution D, where $p_D(x) = \prod_i \Pr_D(x_i)$. On average, problems with skewed example distributions require fewer node expansions than problems with near-uniform distributions.

The fact that the search algorithm tends to perform better with skewed distributions is good news, as there are reasons to believe that the example distributions of real-world problems will tend to be skewed. One reason is simply that it is often the case that some values for a feature are more likely than others, so that examples are more likely to be drawn from the corresponding region of the example space. Even if the possible values of a particular input are equally likely, however, there may be combinations of values for a set of input features that are not possible, such that there would be some regions from which examples would never be drawn. More generally, there are likely to be many correlations between input features that will make some examples much more likely to be seen than others.

Although the true example distributions behind the data sets used in this paper are unknown, input skew can be estimated from the data. Table 2.5 shows the estimated average input skew (Equation 2.34) of each data set, as well as the maximum and minimum input skew ($\max_i skew_D(x_i)$ and $\min_i skew_D(x_i)$, respectively). For all but the Voting data set, the average input skew is near the 0.2-0.35 range, which Figure 2.5 suggests is a good amount of skew (on average) for the search algorithm. Note also that the maximum input skew is

Table 2.5: The average, maximum, and minimum input skew. For all but the Voting data set, the average skew is near the 0.2-0.35 range, which Figure 2.5 suggests is a good amount of skew for the search algorithm.

Data Set	Ave Skew	Max Skew	Min Skew
Pima	0.368	0.497	0.271
WBC1	0.198	0.328	0.118
Voting	0.099	0.357	0.013
Heart	0.191	0.493	0.007
SPECT	0.188	0.369	0.006
German	0.326	0.494	0.087
WBC3	0.213	0.496	0.010
WBC2	0.313	0.495	0.020
Chess	0.317	0.499	0.039

usually very high, showing that most of the data sets have at least one input that is very skewed.

The results with real-world data in Table 2.2 suggest that the branch-and-bound search algorithm will often perform better in practice than the expected performance over randomly selected data sets having a similar average input skew and number of dimensions. (For example, notice the low 9.5 average number of nodes expanded for the similarly-sized WBC1 problem, and the 16.3 average for the larger Voting problem.) One reason for this is that there are factors unrelated to the example distribution that also affect performance (e.g., properties of the spectrum of the function). However, there are also example distribution factors that are not captured by the input skew analysis, such as correlation between inputs. For example, in the Voting data set, although the average input skew is low, many input features are significantly correlated. Therefore, although the probability of an input being 0 or 1 is usually close to 0.5, the probability becomes very skewed when conditioned on the other feature values. Consequently, most examples' inputs are very similar to those of other examples, meaning that they were drawn from similar regions of the input space.

Figure 2.6 demonstrates how correlations can affect performance. There are three plots in Figure 2.6, each showing average node expansions as a function of a skew parameter

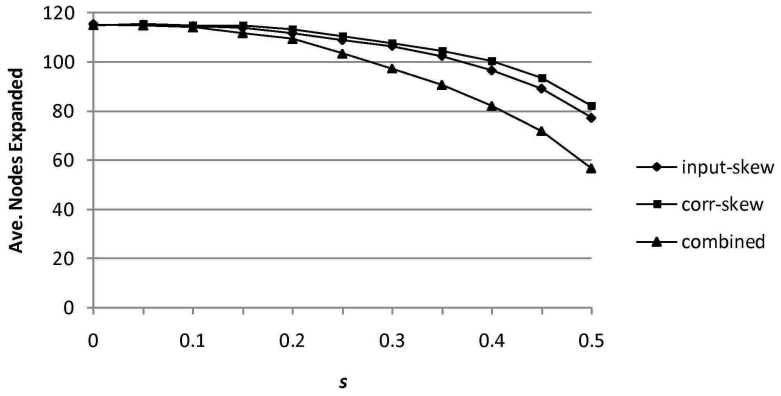


Figure 2.6: The average number of nodes expanded to find the largest XOR coefficient of a 10-dimensional learning problem as a function of an example distribution skew parameter s. For *input-skew*, the distributions are of the form $p_D(x) = \Pr_D(x_1)(\frac{1}{2})^9$, with $\Pr_D(x_i = 1) = \frac{1}{2} + s$. For *corr-skew*, the distributions are of the form $p_D(x) = \Pr_D(x_2|x_1)(\frac{1}{2})^9$, with $\Pr_D(x_2 = 1|x_1 = 1) = 0.5 + s$ and $\Pr_D(x_2 = 1|x_1 = 0) = 0.5 - s$. For *combined*, the distributions are of the form $p_D(x) = \Pr_D(x_2|x_1)\Pr_D(x_1)(\frac{1}{2})^8$, where $\Pr_D(x_1 = 1) = 0.5 + s$, $\Pr_D(x_2 = 1|x_1 = 1) = 0.5 + s$, and $\Pr_D(x_2 = 1|x_1 = 0) = 0.5 - s$. The number of node expansions decreases as input and correlation skew increases, and the reduction is even greater when the two types of skew are combined.

s. One plot, *input-skew*, is a replication of the plot in Figure 2.4, and $s = skew_D(x_1)$ in that case. Another plot, *corr-skew*, shows how performance varies as one input becomes increasingly correlated with another. Here, the distributions are of the form $p_D(x) = \Pr_D(x_2|x_1)\prod_{i:i\neq 2}\Pr_D(x_i) = \Pr_D(x_2|x_1)(\frac{1}{2})^9$, and $\Pr_D(x_2|x_1)$ is a function of s. Specifically, $\Pr_D(x_2 = 1|x_1 = 1) = 0.5 + s$ and $\Pr_D(x_2 = 1|x_1 = 0) = 0.5 - s$. Thus, the values of all inputs but x_2 are assumed to be independent (with no input skew), and the value of x_2 is increasingly likely to match the value of x_1 as s increases. As Figure 2.6 shows, the average node count decreases as x_2 becomes increasingly correlated with x_1. In fact, the reduction in nodes expanded is very similar to the reduction observed when the skew of a single input increases (*input-skew*).

Note that although the distributions that produced the *corr-skew* plot are increasingly skewed as s increases (because examples in which $x_1 = x_2$ become more likely than those in which $x_1 \neq x_2$), the input skew is always 0, even for x_2. ($\Pr_D(x_2 = 1) = \Pr_D(x_2 = 1|x_1 =$

$1)\Pr_D(x_1 = 1) + \Pr_D(x_2 = 1|x_1 = 0)\Pr_D(x_1 = 0) = (0.5 + s)(0.5) + (0.5 - s)(0.5) = 0.5.$)
Thus, these distributions demonstrate how correlation skew can be unrelated to input skew.

The third plot in Figure 2.6 shows how the average number of nodes expanded decreases when the two types of distribution skew are combined. Specifically, it is a plot of node expansions as a function of distributions of the form $p_D(x) = \Pr_D(x_2|x_1) \prod_{i:i\neq 2} \Pr_D(x_i) = \Pr_D(x_2|x_1)\Pr_D(x_1)(\frac{1}{2})^8$, where $\Pr_D(x_1 = 1) = 0.5 + s$, $\Pr_D(x_2 = 1|x_1 = 1) = 0.5 + s$, and $\Pr_D(x_2 = 1|x_1 = 0) = 0.5 - s$. As s increases, the combined effect of the two types of skew is a greater reduction in nodes expanded.

One final important note is that the method for converting non-Boolean inputs to Boolean inputs can affect the skew of an example distribution. Several of the data sets used in this paper contain nominal and/or real-valued input features that were converted to Boolean by the methods described at the beginning of Section 2.7. Nominal input features were replaced by m Boolean features, one for each value of the feature, such that each Boolean input indicated whether the corresponding nominal value was the correct value for that example. Real-valued input features were replaced by a single Boolean value indicating whether the real value was above or below a threshold. Using these methods, the skew of a Boolean input that is derived from a nominal feature depends on the frequency of the corresponding nominal input value. The skew of a Boolean input that is derived from a real-valued feature depends on which threshold is chosen. In spite of the potential differences in skew between features that are originally Boolean and those that are derived Boolean features, an analysis of the data sets used in this paper suggests that the average skews may not differ much in practice. For these data sets, the average skew of input features that were originally Boolean is 0.242, while the average skews for Boolean inputs that were derived from nominal and real-valued features are 0.248 and 0.270, respectively.

2.8.2 Coefficient Size & Distribution

Properties of the spectral representation of the function being learned also affect performance. One of the spectral properties that affects performance is the deviation of the spectrum from a uniform spectrum in which all coefficients have the same magnitude. Using metrics similar to those in the previous section, we will define the skew of a spectral representation $\check{f}$ to be the average absolute difference between each squared coefficient and $\frac{1}{2^n}$:

$$skew(\check{f}) = \frac{1}{2^n} \sum_{\alpha \in \{0,1\}^n} \left| \check{f}(\alpha)^2 - \frac{1}{2^n} \right| \tag{2.35}$$

Note that $\frac{1}{2^n}$ is the squared coefficient size in a uniform spectrum in which $\sum_\alpha \check{f}(\alpha)^2 = 1$. (This choice for $\sum_\alpha \check{f}(\alpha)^2$ is due to the fact that $\sum_\alpha \check{f}(\alpha)^2 = 1$ when $\check{f}$ is the Fourier spectrum of a Boolean function. Squared coefficient size is used instead of absolute coefficient size for computational convenience.) The skew of a single coefficient is the difference between the square of that coefficient and $\frac{1}{2^n}$:

$$skew(\check{f}(\alpha)) = \left| \check{f}(\alpha)^2 - \frac{1}{2^n} \right| \tag{2.36}$$

Figure 2.7 demonstrates how skew in a spectrum can affect search performance. In the figure, the average number of nodes expanded is plotted as a function of $s = skew(\check{f})/(2(1 - \frac{1}{2^n}))$, which expresses the skew of the spectrum as a fraction of the largest possible skew. (The largest possible skew occurs when one coefficient is 1 and the others are 0, in which case the total skew is $(1 - \frac{1}{2^n}) + (2^n - 1)(\frac{1}{2^n}) = (1 - \frac{1}{2^n}) + (1 - \frac{1}{2^n}) = 2(1 - \frac{1}{2^n})$.) As before, results are averaged over 1,000 data sets, but this time a 10-dimensional spectral representation $\check{f}$ with the specified skew is randomly generated (subject to the constraint that $\sum_\alpha \check{f}(\alpha)^2 = 1$), and then $\check{f}$ is transformed (by Equation 2.1) into a function f from which 256 examples are randomly chosen according to a uniform distribution over the example space. As the figure shows, a uniform spectrum is the most difficult case, on average, while the number of nodes

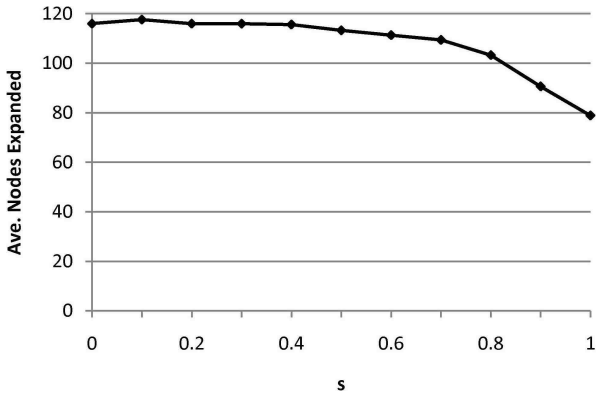


Figure 2.7: The average number of nodes expanded to find the largest XOR coefficient of a 10-dimensional learning problem as a function of $s = skew(\check{f})/(2(1 - \frac{1}{2^n}))$, with a uniform example distribution. On average, fewer node expansions are required for functions with skewed spectral representations.

expanded tends to decrease as the spectrum becomes increasingly skewed. The best result is when $s = 1$, which occurs when there is a single large coefficient and all other coefficients are 0.

Figure 2.8 shows that the node count decreases more rapidly when there is a clearer separation between the largest coefficient and the others. In this figure, average nodes expanded is again plotted as a function of $s = skew(\check{f})/(2(1 - \frac{1}{2^n}))$, but this time there is a single coefficient that is skewed towards 1 while the remaining coefficients have the same magnitude and are shrunk towards 0. The plot in the figure shows that average node count decreases to best-case levels more rapidly. Of course, for the task of finding the single largest coefficient this is a favorable scenario.

Although the spectra of the real-world functions of the data sets used in this paper are unknown, the approximated spectra (Equation 2.26) of the data sets show some significant skews. For the smaller data sets, for which computing the entire spectrum is feasible, the following table shows the skew of the approximated spectrum as a fraction of the maximum possible skew for a data set with the same sum of squared coefficients:

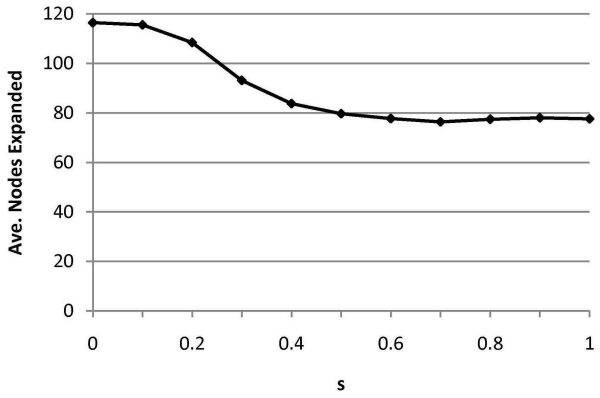


Figure 2.8: The average number of nodes expanded to find the largest XOR coefficient of a 10-dimensional learning problem for which the spectrum has one large coefficient and $2^n - 1$ smaller coefficients of equal size, as a function of $s = skew(\check{f})/(2(1 - \frac{1}{2^n}))$, with a uniform example distribution. The average number of node expansions decreases rapidly as the separation between the large and small coefficients increases.

Data Set	$skew(\check{f}_X)$/MaxPossibleSkew
Pima	0.137
Wisc1	0.135
Voting	0.593
Heart	0.503
SPECT	0.462

The skew in the Voting, Heart, and SPECT data sets is considerably large relative to the maximum possible skew. Even for these data sets, however, a graph like Figure 2.7 for functions of the same dimensionality would likely overestimate the actual node counts. This is in large part due to the fact that these experiments assumed a uniform distribution over the input space. In fact, empirical results suggest that skew in the example distribution has a more significant impact than skew in $\check{f}$. Plotting average node expansions as a function of both example distribution skew and spectrum skew shows that the result is nearly identical to the plot of node expansions vs. example distribution skew when the function is randomly selected (Figure 2.5). In other words, the choice of function/spectrum matters less than the skew in the example distribution.

2.9 Conclusion

Identifying the large coefficients of a spectral representation is a critical problem for spectral learning algorithms. As shown in this paper, for a class of spectral representations that includes the Fourier representation, the general problem of finding large coefficients is NP-Complete. However, this paper presents a method for bounding coefficient size in arbitrary regions of a spectral representation that allows search algorithms to explore the space of coefficients efficiently. Empirical results show that a complete branch-and-bound search algorithm based on this technique can find the largest coefficients quickly, while ignoring most of the exponentially-large search space. In addition, empirical results show that an incomplete beam search algorithm can usually find solutions that are as good as those returned by a complete algorithm, even when the beam is very narrow. Finally, the paper demonstrates how the performance of the branch-and-bound algorithm improves when there is significant non-uniformity in the distribution over examples and/or in the spectrum of the target function.

One interesting direction for future work is to more thoroughly analyze the tradeoffs and benefits of complete and incomplete approaches to finding large coefficients. For example, Figure 2.4 shows that the beam search algorithm can use a very narrow beam and still find a basis function that is comparable, in terms of test accuracy, to the basis function with largest coefficient. However, how do learning accuracies compare after several basis functions are added? This comparison may be particularly interesting in the context of boosting, as incomplete algorithms may be able to be very greedy in that scenario and still match the performance obtained by a complete search.

Also, although Section 2.8 discusses some properties of learning problems that affect the performance of the search algorithms, the results on randomly generated problems having similar characteristics (based on these properties) tend to be harder on average than similarly-sized real-world problems. Therefore, one area of future work is to further identify

properties of learning problems that affect performance, so there can be a better understanding of when it will or will not be easy to find large coefficients in real-world settings.

2.10 Appendix: Additional Theorems

Theorem 8. *Let X be a set of $\langle x, f(x)\rangle$ examples, with $x \in \{0,1\}^n$ and $f(x) \in \mathbb{R}$, and let ϕ_α be a function of the form $\phi_\alpha : \{0,1\}^n \to \mathbb{R}$. Then*

$$\min_{\hat{g}_X(\alpha)} \sum_{\langle x, f(x)\rangle \in X} \left(f(x) - \hat{g}_X(\alpha)\phi_\alpha(x)\right)^2 = \frac{1}{\|\phi_\alpha\|_X^2} \sum_{\langle x, f(x)\rangle \in X} f(x)\phi_\alpha(x)$$

Proof. Assume there is an ordering over X such that each example can be identifed by in index i, where $0 \leq i < |X|$, and let $\vec{x}$ and $\vec{y}$ be vectors of length $|X|$ such that $\forall i((\langle x, f(x)\rangle_i \in X) \Rightarrow (\vec{x}_i = \phi_\alpha(x) \wedge \vec{y}_i = f(x)))$. Then the coefficient $\hat{g}_X(\alpha)$ that minimizes $\sum_{\langle x, f(x)\rangle \in X} \left(f(x) - \hat{g}_X(\alpha)\phi_\alpha(x)\right)^2$ is the coefficient c that minimizes $\sum_i (\vec{y}_i - c\vec{x}_i)^2$. c is given by the least squares formula $c = (\vec{x}^T\vec{x})^{-1}\vec{x}^T\vec{y}$, so $\hat{g}_X(\alpha)$ is given by:

$$\begin{aligned}
\hat{g}_X(\alpha) = c &= (\vec{x}^T\vec{x})^{-1}\vec{x}^T\vec{y} \\
&= \left(\sum_{\langle x, f(x)\rangle \in X} \phi_\alpha(x) \cdot \phi_\alpha(x)\right)^{-1} \vec{x}^T\vec{y} \\
&= \left(\|\phi_\alpha\|_X^2\right)^{-1} \vec{x}^T\vec{y} \\
&= \frac{1}{\|\phi_\alpha\|_X^2} \left(\vec{x}^T\vec{y}\right) \\
&= \frac{1}{\|\phi_\alpha\|_X^2} \sum_{\langle x, f(x)\rangle \in X} f(x)\phi_\alpha(x)
\end{aligned}$$

□

Theorem 9. *If f is of the form $f : \{0,1\}^n \to \mathbb{R}$ and $B = \{\phi_\alpha : \alpha \in \{0,1\}^n\}$ is a basis for f, then f can be expressed in B^{-1} as*

$$f(x) = \sum_{\alpha \in \{0,1\}^n} \|\phi_\alpha\|^2 \check{f}(\alpha) \phi_x^{-1}(\alpha)$$

Proof. From the definition of $\check{f}$ (Equation 2.23),

$$\begin{aligned}
\sum_{\alpha \in \{0,1\}^n} \|\phi_\alpha\|^2 \check{f}(\alpha) \phi_x^{-1}(\alpha) &= \sum_{\alpha \in \{0,1\}^n} \|\phi_\alpha\|^2 \left(\frac{1}{\|\phi_\alpha\|^2} \sum_{x' \in \{0,1\}^n} f(x') \phi_\alpha(x') \right) \phi_x^{-1}(\alpha) \\
&= \sum_{\alpha \in \{0,1\}^n} \left(\sum_{x' \in \{0,1\}^n} f(x') \phi_\alpha(x') \right) \phi_x^{-1}(\alpha) \\
&= \sum_{\alpha \in \{0,1\}^n} \sum_{x' \in \{0,1\}^n} f(x') \phi_\alpha(x') \phi_x^{-1}(\alpha) \\
&= \sum_{x' \in \{0,1\}^n} \sum_{\alpha \in \{0,1\}^n} f(x') \phi_\alpha(x') \phi_x^{-1}(\alpha) \\
&= \sum_{x' \in \{0,1\}^n} f(x') \sum_{\alpha \in \{0,1\}^n} \phi_\alpha(x') \phi_x^{-1}(\alpha)
\end{aligned}$$

And, from the definition of an inverse basis, $\sum_{\alpha \in \{0,1\}^n} \phi_\alpha(x') \phi_x^{-1}(\alpha) = 1$ if $x' = x$, and $\sum_{\alpha \in \{0,1\}^n} \phi_\alpha(x') \phi_x^{-1}(\alpha) = 0$ otherwise. Therefore,

$$\sum_{x' \in \{0,1\}^n} f(x') \sum_{\alpha \in \{0,1\}^n} \phi_\alpha(x') \phi_x^{-1}(\alpha) = f(x)$$

□

Chapter 3

An Empirical Comparison of Spectral Learning Methods for Classification

Abstract

In this paper, we explore the problem of how to learn spectral (e.g., Fourier) models for classification problems. Specifically, we consider two sub-problems of spectral learning: (1) how to select the basis functions that will be included in the model and (2) how to assign coefficients to the selected basis functions. Empirical results show that a method for assigning coefficients based on finding an optimal linear combination of selected basis functions usually outperforms the traditional approach (estimating coefficients from data), while a low-order approach to basis function selection usually outperforms other basis function selection methods.

3.1 Introduction

Spectral learning methods based on Fourier, wavelet, and other transforms have been successfully applied in both applied and theoretical domains [Donoho and Johnstone, 1994, 1995, Drake and Ventura, 2005, 2009, Jackson, 1997, Kargupta et al., 1999, Kushilevitz and Mansour, 1993, Linial et al., 1993, Mansour and Sahar, 2000]. The common theme of these approaches is the end goal of representing the target function in a particular spectral representation. However, several different approaches to spectral learning have been used, and it is not clear which are most effective in typical machine learning scenarios.

In this paper, we explore the problem of how to best learn spectral representations for classification problems. In doing so, we compare and analyze new and old approaches to the two main phases of the spectral learning process: determining which basis functions to include in the model and determining the coefficients to assign to each basis function.

3.2 Background

Spectral representations provide an alternative representation of a function. For example, consider the Fourier spectrum. Suppose f is a real-valued function of n Boolean inputs (i.e., $f : \{0,1\}^n \longrightarrow \mathbb{R}$). Then the Fourier spectrum of f, denoted $\hat{f}$, is given by

$$\hat{f}(\alpha) = \frac{1}{2^n} \sum_{x \in \{0,1\}^n} f(x)\chi_\alpha(x) \tag{3.1}$$

where $\alpha \in \{0,1\}^n$ is the label of basis function χ_α, which is defined by

$$\chi_\alpha(x) = \begin{cases} +1 : \text{if } \sum_i \alpha_i x_i \text{ is even} \\ -1 : \text{if } \sum_i \alpha_i x_i \text{ is odd} \end{cases} \tag{3.2}$$

in which α_i and x_i denote the i^{th} binary digits of α and x. Each Fourier coefficient $\hat{f}(\alpha)$ corresponds to a particular basis function, χ_α, and the sign and magnitude of $\hat{f}(\alpha)$ indicate

the correlation between f and χ_α. Large positive and negative coefficients indicate significant positive and negative correlations, respectively, while small coefficients indicate little or no correlation.

Any f can be recovered from its Fourier representation by applying the inverse transform:

$$f(x) = \sum_{\alpha \in \{0,1\}^n} \hat{f}(\alpha)\chi_\alpha(x) \tag{3.3}$$

Equation 3.3 also reveals that the Fourier spectrum provides a representation of f as a linear combination of the Fourier basis functions.

In the case of an n-dimensional Boolean-input function, the Fourier basis functions are XOR functions, each returning -1 iff the XOR of a particular subset of the inputs is true. The subset is implicitly defined by α in Equation 3.2. Since $\alpha_i x_i = 0$ when $\alpha_i = 0$ and $\alpha_i x_i = x_i$ when $\alpha_i = 1$, the output of χ_α depends only those inputs for which $\alpha_i = 1$. The order of any χ_α is given by $\sum_i \alpha_i$, which is the number of inputs that are relevant to χ_α.

By changing the basis function definition so that logical ANDs and ORs are computed instead of XORs, we obtain new bases. Below are definitions of AND (ξ) and OR (ζ) basis functions:

$$\xi_\alpha(x) = \begin{cases} +1 \ : \text{ if } \ \sum_i \alpha_i x_i < \sum_i \alpha_i \\ -1 \ : \text{ if } \ \sum_i \alpha_i x_i = \sum_i \alpha_i \end{cases} \tag{3.4}$$

$$\zeta_\alpha(x) = \begin{cases} +1 \ : \text{ if } \ \sum_i \alpha_i x_i = 0 \\ -1 \ : \text{ if } \ \sum_i \alpha_i x_i > 0 \end{cases} \tag{3.5}$$

By replacing the Fourier basis functions in Equation 3.1 with either of these sets of basis functions, we obtain new "correlation spectra." That is, the coefficients will reveal the correlation between f and either the AND or OR functions, just as the Fourier coefficients do for the XOR functions. Note, however, that unlike in the XOR case, the coefficients obtained from Equation 3.1 will not generally give the linear combination of AND or OR functions

that equals f. There is another transform equation that gives the linear combination (but not the correlation); however, only the correlation spectrum will be of interest here.

3.3 Spectral Learning Methods

Given a set X of $\langle x, f(x)\rangle$ examples, a spectral learning algorithm attempts to learn a spectral representation of f that approximates it well. Since the number of basis functions is exponential in the number of inputs to a function, a spectral learning algorithm will typically select a subset of basis functions to use in its model, implicitly assigning coefficients of 0 to the remaining basis functions. If A is the set of labels of basis functions included in the model, then a spectral learner's approximation of f is given by the following:

$$\tilde{f}(x) = \sum_{\alpha \in A} \hat{f}(\alpha)\phi_\alpha(x) \tag{3.6}$$

where ϕ_α is a general basis function reference that could be replaced by any of the basis functions defined in the previous section.

Spectral learning algorithms can be applied to Boolean classification problems by encoding the outputs of positive and negative examples as -1.0 and 1.0, respectively, and using the sign of the model's output to make classifications:

$$\tilde{f}(x) = \begin{cases} false & : \text{if } \sum_{\alpha \in A} \hat{f}(\alpha)\phi_\alpha(x) \geq 0 \\ true & : \text{if } \sum_{\alpha \in A} \hat{f}(\alpha)\phi_\alpha(x) < 0 \end{cases} \tag{3.7}$$

The task of a spectral learner is to determine which basis functions to include in the model and what coefficient values to assign to each basis function.

3.3.1 Selecting Basis Functions

Three approaches to basis function selection are considered in this paper: Most-Correlated, Low-Order, and AdaBoost.

Most-Correlated

The most common approach to basis function selection is to select the basis functions that are most correlated with f, or, equivalently, that have the largest coefficients in the correlation spectrum of f (Equation 3.1) [Blum et al., 1994, Donoho and Johnstone, 1994, 1995, Drake and Ventura, 2005, Kushilevitz and Mansour, 1993, Mansour and Sahar, 2000]. Although the true coefficients are unknown, they can be estimated from X:

$$\tilde{\hat{f}}(\alpha) = \frac{1}{|X|} \sum_{\langle x, f(x) \rangle \in X} f(x)\phi_\alpha(x) \tag{3.8}$$

Stated precisely, the Most-Correlated selection method used in this paper selects basis functions according to the following rule:

$$\begin{aligned} &\phi_\alpha \text{ is preferred to } \phi_\beta \text{ iff:} \\ &\quad (|\hat{f}(\alpha)| > |\hat{f}(\beta)|) \vee \\ &\qquad (|\hat{f}(\alpha)| = |\hat{f}(\beta)| \wedge \textstyle\sum_i \alpha_i < \sum_i \beta_i) \end{aligned}$$

Note that ties in coefficient size are broken in favor of lower-order basis functions. (If there is still a tie, it is broken randomly.)

For any basis, the Most-Correlated approach makes sense from a feature selection perspective, as basis functions that are correlated with f should tend to be better features. For a basis in which the correlation spectrum gives the representation of a function in that basis, such as the Fourier spectrum, this approach can also be motivated by the goal of trying to approximate the true representation of f in that basis, because then the sensible strategy would be to select the basis functions with large coefficients, as they carry the most "weight" in the linear combination.

Low-Order

Another reasonable approach to basis function selection is to use the low-order basis functions (e.g., to select all basis functions for which $\sum_i \alpha_i \leq k$) [Bshouty and Tamon, 1996, Linial

et al., 1993]. The Low-Order approach used in this paper selects basis functions according to the following rule:

$$\begin{aligned}&\phi_\alpha \text{ is preferred to } \phi_\beta \text{ iff:}\\ &\quad(\textstyle\sum_i \alpha_i < \sum_i \beta_i) \vee\\ &\qquad(\textstyle\sum_i \alpha_i = \sum_i \beta_i \wedge |\hat{f}(\alpha)| > |\hat{f}(\beta)|)\end{aligned}$$

Note that basis functions of the same order are selected in order of highest correlation with the training data. (If there is still a tie, it is broken randomly.) Thus, the Low-Order and Most-Correlated methods both favor low-order functions and high correlations, and they differ only in which criterion is considered more important.

One motivation for preferring low-order functions is that it seems reasonable to expect that in practice f is more likely to be correlated with lower-order functions. (In fact, a low-order approach can be viewed as a most-correlated approach with the prior assumption that lower-order functions will be more correlated with f.) Low-order functions may be more correlated with f because they are defined over fewer inputs and therefore represent a simpler interaction between inputs. It also seems reasonable to expect that lower-order functions will be more likely to generalize well.

AdaBoost

An alternative approach to basis function selection is to select basis functions in conjunction with a boosting algorithm [Jackson, 1997]. A boosting algorithm generates an ensemble of learners that makes classifications by weighted vote. The learners are trained iteratively, typically with the first learner trained on the original data set and subsequent learners trained on weighted data sets in which examples that were misclassified by previously-trained learners receive more weight. If the learners are spectral learners whose models consist of a single basis function, then the result is just a spectral representation in which the basis functions (and possibly coefficients) were selected in conjunction with a boosting algorithm.

SelectBasisFunctions–AdaBoost(X, T)

(1) $A \leftarrow \emptyset$
(2) for each $\langle x, f(x)\rangle \in X$
(3) $w_{\langle x,f(x)\rangle} \leftarrow \frac{1}{|X|}$
(4) for $t = 1$ to T
(5) $X_t \leftarrow \{\langle x, w_{\langle x,f(x)\rangle} f(x)\rangle : \langle x, f(x)\rangle \in X\}$
(6) $\phi_{\alpha_t} \leftarrow$ SelectCorrelatedFunction(X_t)
(7) $A \leftarrow A \cup \phi_\alpha$
(8) $\epsilon_t \leftarrow \sum_{\{\langle x,f(x)\rangle : \phi_{\alpha_t}(x) \neq f(x)\}} w_{\langle x,f(x)\rangle}$
(9) for each $\langle x, f(x)\rangle \in X$ s.t. $\phi_{\alpha_t}(x) = f(x)$
(10) $w_{\langle x,f(x)\rangle} \leftarrow w_{\langle x,f(x)\rangle}(\frac{\epsilon_t}{1-\epsilon_t})$
(11) $z \leftarrow \sum_{\langle x,f(x)\rangle \in X} w_{\langle x,f(x)\rangle}$
(12) for each $\langle x, f(x)\rangle \in X$
(13) $w_{\langle x,f(x)\rangle} \leftarrow w_{\langle x,f(x)\rangle}/z$
(14) return A

Figure 3.1: The AdaBoost basis function selection procedure.

The AdaBoost basis function selection approach used in this paper is based on the AdaBoost.M1 algorithm [Freund and Schapire, 1996], and is illustrated in Figure 3.1. In each boosting iteration t, the weights for each example are used to create a weighted data set, X_t (line 5), in which the weights are implicitly represented by converting the $f(x)$ values from ± 1 to $\pm w_{\langle x,f(x)\rangle}$. Then, a basis function that is highly correlated with X_t is selected (line 6) and added to the solution (line 7). The distribution of weight over the examples is initially uniform (line 3), but it is updated each iteration (lines 8-13) so that examples that would be classified correctly by the most recently added basis function receive less weight (lines 9-10). (Note: For simplicity, the algorithm in Figure 3.1 is presented as if each ϕ_{α_t} is positively correlated with X_t. However, if ϕ_{α_t} is negatively correlated, each occurrence of $\phi_{\alpha_t}(x)$ in lines 8 and 9 should be replaced with $(-\phi_{\alpha_t}(x))$.)

3.3.2 Assigning Coefficients

Three methods of assigning coefficients to selected basis functions are considered in this paper: Data-Estimate, Min-Squared-Error, and AdaBoost.

Data-Estimate

The Data-Estimate approach, or some variation of it, is by far the most common method for assigning coefficients to basis functions [Blum et al., 1994, Bshouty and Tamon, 1996, Donoho and Johnstone, 1994, 1995, Kushilevitz and Mansour, 1993, Linial et al., 1993, Mansour and Sahar, 2000]. In its basic form, each basis function is assigned the coefficient that is estimated from training data. Typically, this is done by Equation 3.8, which is also the method used here.

If the goal is to approximate the true spectral representation of f, then setting each coefficient to the value estimated from the training data would seem to be a natural choice, especially if the basis function selection approach is motivated by the same goal. Regardless of how basis functions are selected, however, the Data-Estimate method can be reasonably motivated as an ensemble-building approach that weights each basis function in proportion to its classification accuracy over X.

Min-Squared-Error

The Min-Squared-Error coefficient assignment method can be motivated by viewing spectral learning from a feature selection perspective. From this perspective, basis function selection can be thought of as the task of identifying a good set of features, while coefficient assignment can be thought of as the task of learning an "optimal" linear combination of the features, without regard to whether the resulting combination resembles the true spectral representation of the function. In the Min-Squared-Error approach, the optimal linear combination of the set A of selected basis functions is the one that minimizes the squared error over the training data:

$$\underset{\tilde{\hat{f}}(\alpha_1),\ldots,\tilde{\hat{f}}(\alpha_{|A|})}{\operatorname{argmin}} \left(\sum_{\langle x, f(x)\rangle \in X} \left(f(x) - \sum_{\alpha \in A} \tilde{\hat{f}}(\alpha)\phi_\alpha(x) \right)^2 \right)$$

Motivations for using squared error as the metric for optimality in the linear combination include the fact that it generalizes naturally to regression problems and that it is easily

computed. Other metrics, such as the number of misclassifications or the distance from the decision surface to the nearest training examples of each class, could also be used, but are not considered here.

Note that there may not be a unique solution to the least-squares problem, indicating that with respect to the data there is redundancy in the set of selected basis functions. To resolve this issue, basis functions are considered for inclusion in the model iteratively (either in the order defined by the preference function or the order in which they were added to the model by AdaBoost), and any basis functions whose inclusion would introduce redundancy are not added to the model.

AdaBoost

The AdaBoost coefficient assignment method makes sense only in the context of the AdaBoost basis function selection method. In the AdaBoost.M1 algorithm, each learner is assigned a coefficient whose magnitude is proportional to the learner's accuracy on its weighted set of training data. In terms of the AdaBoost basis function selection method described in Figure 3.1, each coefficient is given by the following:

$$\hat{f}(\alpha_t) = \pm \log\left(\frac{1-\epsilon_t}{\epsilon_t}\right) \tag{3.9}$$

where ϵ_t is the (weighted) misclassification rate of ϕ_{α_t} and the sign of $\hat{f}(\alpha_t)$ is negative iff $\sum_{\langle x, f(x)\rangle \in X_t} f(x)\phi_{\alpha_t}(x) < 0$ (i.e., if ϕ_{α_t} is negatively correlated with X_t).

When using AdaBoost to select basis functions, it seems natural to use AdaBoost to assign coefficients. However, we will consider the possibility that there may be a superior method of assigning coefficients, even if basis functions were selected by AdaBoost.

3.4 Empirical Results

In this section, the basis function selection and coefficient assignment methods are compared on nine learning problems [Newman et al., 1998] with each of the previously defined spectral representations (i.e., the AND, OR, and XOR bases). For each learning problem, the data was partitioned 100 times into training and test sets (with 10% used for testing), and the average classification accuracy on the test set when training on the corresponding training set was recorded. Each method used the same 100 splits of data, and results were averaged over those 100 trials. Statistically significant differences were measured pair-wise by a paired permutation test, with significant differences defined as those for which $p \leq 0.01$.

For each spectral learning approach there is a single free parameter: T, the number of basis functions to include in the model. This parameter was set automatically as part of the learning process. Specifically, each learner would split its training data into training and validation sets (with 10% held out for validation), and would then estimate its generalization performance with each number of basis functions from 1 to T_{max}. After repeating this on 10 random partitions of the training data and averaging results, T was set to the number of basis functions that maximized classification accuracy on the validation set (with ties broken in favor of fewer basis functions). Then, the learner would train on the entire training set with the selected T value.

3.4.1 Assigning Coefficients

Tables 3.1 and 3.2 show the average test accuracy when using the Data-Estimate and Min-Squared-Error coefficient assignment methods with the Low-Order and Most-Correlated basis function selection approaches. Of the 27 possible combinations of data set and basis, the tables show only those cases for which a statistically significant difference between methods was observed. In each case, the higher accuracy is highlighted in bold. Where there were significant differences, the Min-Squared-Error approach is usually superior to the Data-Estimate approach.

Table 3.1: Comparison of coefficient assignment methods for the Low-Order basis function selection approach. Where significant differences were observed (shown below), the Min-Squared-Error approach is usually superior.

Data Set	Basis	DataEst	MinSqErr
Chess	AND	87.2%	**95.2%**
Chess	OR	86.8%	**96.6%**
Chess	XOR	87.0%	**94.9%**
German	AND	69.9%	**73.1%**
German	OR	70.8%	**73.7%**
German	XOR	70.4%	**73.2%**
Pima	XOR	72.6%	**73.5%**
SPECT	AND	79.3%	**81.9%**
SPECT	OR	79.0%	**81.9%**
SPECT	XOR	77.3%	**81.6%**
Voting	AND	95.4%	**95.8%**
Voting	XOR	95.3%	**95.9%**
Wisc1	AND	95.5%	**96.0%**
Wisc1	OR	**96.3%**	95.8%
Wisc1	XOR	95.6%	**96.2%**
Wisc2	AND	**75.5%**	71.5%
Wisc2	OR	**75.6%**	72.4%
Wisc2	XOR	**76.3%**	72.6%
Wisc3	AND	91.3%	**94.3%**

One advantage of the Min-Squared-Error approach is that it has more flexibility in modeling functions. In the Data-Estimate approach, the coefficients are assigned independently, without regard to what other basis functions may be in the model. In the Min-Squared-Error approach, meanwhile, the coefficients are set as a group to be "optimal" with respect to the selected set of basis functions. Of course, the minimum squared error linear combination is not certain to be better, and increased flexibility can increase the likelihood of overfitting, which may explain why the Min-Squared-Error approach occasionally performed worse. However, it seems to be a better approach in general.

Table 3.3 shows a comparison of the AdaBoost, Data-Estimate, and Min-Squared-Error coefficient assignment methods when AdaBoost is used to select basis functions. In general, the AdaBoost coefficient assignment method gives the best results.

Table 3.2: Comparison of coefficient assignment methods for the Most-Correlated basis function selection approach. Where significant differences were observed (shown below), the Min-Squared-Error approach is usually superior.

Data Set	Basis	DataEst	MinSqErr
Chess	AND	80.7%	**81.0%**
Chess	OR	77.6%	**89.0%**
Chess	XOR	75.5%	**83.2%**
German	AND	69.6%	**70.9%**
Heart	AND	79.2%	**81.5%**
Pima	AND	**74.1%**	73.3%
Pima	OR	72.5%	**73.6%**
SPECT	OR	**83.7%**	82.6%
Voting	AND	95.5%	**95.8%**
Wisc1	AND	93.2%	**96.0%**
Wisc3	AND	90.8%	**92.8%**

Since both the Data-Estimate and AdaBoost methods assign coefficients that are proportional to classification accuracy, with the primary difference being whether accuracy is measured with respect to the original data or a weighted version of the data, it may be surprising that AdaBoost gave a significantly better result so often. However, an important difference is that while the Data-Estimate coefficients are assigned independently, the iteratively-assigned AdaBoost coefficients are each dependent on previously-added basis functions. In AdaBoost, the weighted data sets implicitly carry information about previously added basis functions, which allows the assigned coefficients to be "optimized" in a sense with respect to previously added basis functions. As with the Min-Squared-Error approach, however, this extra flexibility may lead to overfitting in some cases.

There were few significant differences between the Min-Squared-Error and AdaBoost coefficient assignment methods. Interestingly, however, in those cases where there was a difference, the AdaBoost method is always superior.

Table 3.3: Comparison of coefficient assignment methods for the AdaBoost basis function selection approach. Where significant differences were observed (shown below), the AdaBoost coefficient assignment method is usually superior to the other approaches.

Data Set	Basis	AdaBoost	DataEst
Chess	AND	**97.6%**	94.0%
Chess	OR	**96.1%**	93.9%
Chess	XOR	**97.7%**	94.8%
German	AND	**72.7%**	71.7%
German	OR	**72.6%**	71.4%
Heart	AND	**81.3%**	79.1%
Heart	OR	**81.5%**	78.2%
SPECT	AND	**83.5%**	78.7%
SPECT	OR	82.3%	**83.9%**
Wisc1	AND	**95.9%**	92.8%
Wisc3	AND	**93.9%**	92.7%
Wisc3	OR	**94.8%**	92.8%
Wisc3	XOR	91.6%	**92.8%**

Data Set	Basis	AdaBoost	MinSqErr
Chess	AND	**97.6%**	95.8%
Chess	OR	**96.1%**	95.4%
Chess	XOR	**97.7%**	96.4%
Heart	AND	**81.3%**	79.4%
Wisc1	OR	**96.0%**	95.5%

3.4.2 Selecting Basis Functions

Tables 3.4, 3.5, and 3.6 provide pair-wise comparisons of the AdaBoost, Low-Order, and Most-Correlated basis function selection methods when each is combined with its preferred coefficient assignment method (i.e., Min-Squared-Error coefficient assignment for the Low-Order and Most-Correlated selection methods, and AdaBoost coefficient assignment for the AdaBoost selection method). Again, only cases for which there was a statistically significant difference between methods are shown, and the higher accuracy in each case is highlighted in bold.

Table 3.4 reveals a clear superiority of the Low-Order approach over the Most-Correlated approach. This result is interesting, as the Most-Correlated approach would seem to have an advantage: it can select correlated basis functions from any part of the spectrum,

Table 3.4: Comparison of the Low-Order and Most-Correlated basis function selection methods. Where significant differences were observed (shown below), the Low-Order approach is consistently superior.

Data Set	Basis	Low-Order	Most-Corr
Chess	AND	**95.2%**	81.0%
Chess	OR	**96.6%**	89.0%
Chess	XOR	**94.9%**	83.2%
German	AND	**73.1%**	70.9%
German	OR	**73.7%**	70.1%
German	XOR	**73.2%**	71.6%
Heart	OR	**83.2%**	78.4%
SPECT	AND	**81.9%**	77.9%
SPECT	XOR	**81.6%**	78.0%
Voting	XOR	**95.9%**	95.4%
Wisc1	OR	**95.8%**	94.9%
Wisc2	AND	71.5%	**73.8%**
Wisc3	AND	**94.3%**	92.8%
Wisc3	OR	**94.4%**	92.8%
Wisc3	XOR	**94.0%**	91.8%

while the Low-Order method is restricted to low-order functions. As mentioned previously, it seems reasonable to expect that low-order functions will typically exhibit higher correlation. Thus, we might expect that the two approaches would perform similarly, as both would tend to select low-order functions. However, the results indicate that the Most-Correlated approach often selects high-order functions that are individually more correlated with X than many lower-order functions, but the end result is a set of functions that is collectively less correlated with f. Two possible reasons for this are (1) that the higher-order functions do not generalize as well or (2) that the most correlated basis functions are less effective as a set of features than other sets of basis functions.

Further analysis points to the second reason. Empirical results show that although low-order basis functions tend to be more correlated with X (and f) on average, the error in training-data estimates of correlation do not seem to be worse for high-order basis functions than low-order basis functions. Thus, the highly correlated high-order basis functions may not necessarily be bad features. However, the results in Table 3.7 suggest one problem with

Table 3.5: Comparison of the AdaBoost and Most-Correlated basis function selection methods. Where significant differences were observed (shown below), the AdaBoost approach is usually superior.

Data Set	Basis	AdaBoost	Most-Corr
Chess	AND	**97.6%**	81.0%
Chess	OR	**96.1%**	89.0%
Chess	XOR	**97.7%**	83.2%
German	AND	**72.7%**	70.9%
German	OR	**72.6%**	70.1%
Heart	OR	**81.5%**	78.4%
Heart	XOR	79.3%	**82.7%**
Pima	AND	**74.0%**	73.3%
Pima	OR	72.3%	**73.6%**
SPECT	AND	**83.5%**	77.9%
Wisc1	OR	**96.0%**	94.9%
Wisc1	XOR	94.8%	**95.9%**
Wisc3	AND	**93.9%**	92.8%
Wisc3	OR	**94.8%**	92.8%

the Most-Correlated approach. For each data set and basis combination shown in Table 3.4, Table 3.7 shows the average correlation, measured over the training data, between the first 10 basis functions selected by each method. The results indicate that the most correlated basis functions tend to also be very correlated with each other. Consequently, it is likely that although the Most-Correlated approach picks basis functions that are highly correlated with X (and probably f), they may tend to be so correlated with each other that there is little gained by combining them.

The performance advantage of the AdaBoost approach over the Most-Correlated approach (Table 3.5) is also interesting, and may also be explained in part by Table 3.7. Both methods are based on the idea of choosing correlated basis functions. However, the AdaBoost approach selects basis functions that are correlated with weighted data sets that focus on misclassified examples. This naturally de-correlates the selected functions and results in a set of basis functions that are correlated with different regions of f, rather than all being globally correlated with f.

Table 3.6: Comparison of the AdaBoost and Low-Order basis function selection methods. Where significant differences were observed (shown below), the Low-Order approach is usually superior.

Data Set	Basis	AdaBoost	Low-Order
Chess	AND	**97.6%**	95.2%
Chess	OR	96.1%	**96.6%**
Chess	XOR	**97.7%**	94.9%
German	OR	72.6%	**73.7%**
German	XOR	71.8%	**73.2%**
Heart	OR	81.5%	**83.2%**
Heart	XOR	79.3%	**83.3%**
Pima	OR	72.3%	**73.4%**
SPECT	AND	**83.5%**	81.9%
SPECT	XOR	78.7%	**81.6%**
Voting	XOR	95.3%	**95.9%**
Wisc1	XOR	94.8%	**96.2%**
Wisc3	XOR	91.6%	**94.0%**

Finally, Table 3.6 shows that the Low-Order approach usually outperforms the AdaBoost approach when there is a significant difference. Thus, the Low-Order approach seems to be the best approach overall. However, the AdaBoost approach gave the best result on two of the nine data sets (Chess and SPECT), and may therefore be worth considering. The Most-Correlated approach, on the other hand, would seem to be less useful to consider, as it is often worse than the others and it was never significantly better than both of the others.

3.5 Conclusion

Spectral approaches to machine learning have been successfully applied in several domains, and different methods for learning spectral representations have been proposed. In this paper, we have compared the fundamental approaches to selecting basis functions and assigning coefficients. Interestingly, empirical results suggest that the spectral learning approach that is most common, selecting the most correlated basis functions and estimating their coefficients from data, which is motivated by a desire to estimate the function's true spectral representation, may be the worst approach for typical machine learning problems. On the

Table 3.7: Average correlation (over the training data) between the first 10 basis functions selected by each selection method. The Most-Correlated approach tends to select basis functions that are highly correlated with each other, which can hamper learning.

Data Set	Basis	AdaBoost	Low-Ord	Most-Corr
Chess	AND	0.232	0.242	0.996
Chess	OR	0.356	0.242	0.484
Chess	XOR	0.156	0.240	0.992
German	AND	0.332	0.358	0.794
German	OR	0.292	0.364	0.892
German	XOR	0.106	0.322	0.818
Heart	OR	0.256	0.270	0.720
SPECT	AND	0.382	0.292	0.566
SPECT	XOR	0.104	0.286	0.290
Voting	XOR	0.184	0.498	0.622
Wisc1	OR	0.644	0.614	0.914
Wisc2	AND	0.408	0.404	0.942
Wisc3	AND	0.488	0.710	0.862
Wisc3	OR	0.472	0.704	0.994
Wisc3	XOR	0.268	0.688	0.916

other hand, attempting to learn an optimal linear combination of low-order basis functions appears to be a more effective approach.

Although the results presented in this paper suggest that a spectral learner will perform better if it limits itself to low-order basis functions (even if there are higher-order basis functions that appear to be more highly correlated), not all functions can be approximated well by only low-order basis functions, and it seems reasonable to expect that in some cases a spectral learner would perform better if it could use some useful higher-order basis functions. An important direction for future work will be to determine how to recognize and take advantage of useful higher-order basis without increasing the likelihood of overfitting to training data and losing the good generalization performance of a low-order approach.

Chapter 4

Improving Spectral Learning by Using Multiple Representations

Abstract

Spectral learning algorithms learn an unknown function by learning a spectral (e.g., Fourier) representation of the function. However, there are many possible spectral representations, none of which will be best in all situations. This paper presents and compares methods for learning from multiple spectral representations. Empirical results suggest that an ensemble approach to multi-spectrum learning, in which spectral models are learned independently in each of a set of candidate representations and then combined in a majority-vote ensemble, works best in practice.

4.1 Introduction

Spectral learning algorithms, such as those based on Fourier or wavelet transforms, learn an unknown function f by learning a spectral representation of f from a set of examples. They have been successfully applied in both learning theory and real-world applications [Donoho and Johnstone, 1994, 1995, Drake and Ventura, 2005, 2009, Kushilevitz and Mansour, 1993, Kargupta et al., 2000, Linial et al., 1993, Mansour and Sahar, 2000].

Typically, a particular representation, such as the Fourier or wavelet representation, is chosen for a spectral learner prior to training. However, it is easy to show that any given spectral representation will be effective for some functions and ineffective for others. Consequently, it seems natural to consider spectral learning methods that allow a learner to choose from or combine multiple spectral representations. This paper proposes and compares methods for learning from multiple spectral representations.

4.2 Background

In the following sections, spectral learning is analyzed in the context of Boolean classification problems with Boolean input features. Specifically, the functions being learned are of the form $f : \{0,1\}^n \longrightarrow \{+1,-1\}$. (In many cases the problems have non-Boolean input features that are converted to Boolean features, as described later.)

4.2.1 Spectral Representations

In a spectral representation, a function is expressed in terms of a set of basis functions. In this paper, three spectral representations are considered, each based on a different set of basis functions. One is the Fourier representation, which in the case of Boolean input features is based on XOR basis functions. Specifically, the basis functions of the Fourier/XOR

representation are given by

$$\chi_\alpha(x) = \begin{cases} +1 \ : \text{ if } \ \sum_i \alpha_i x_i \ \text{ is even} \\ -1 \ : \text{ if } \ \sum_i \alpha_i x_i \ \text{ is odd} \end{cases} \tag{4.1}$$

where $\alpha \in \{0,1\}^n$ is the n-digit label of basis function χ_α, and α_i and x_i denote the i^{th} binary digits or inputs of α and x, respectively. Each χ_α returns -1 iff the XOR of a particular subset of the inputs is true. The subset is implicitly defined by α. Since $\alpha_i x_i = 0$ when $\alpha_i = 0$, and since $\alpha_i x_i = x_i$ when $\alpha_i = 1$, the output of χ_α depends only on those inputs for which $\alpha_i = 1$.

The Fourier spectrum of f, denoted $\hat{f}$, is given by

$$\hat{f}(\alpha) = \frac{1}{2^n} \sum_{x \in \{0,1\}^n} f(x)\chi_\alpha(x) \tag{4.2}$$

Each Fourier coefficient $\hat{f}(\alpha)$ corresponds to a particular basis function, χ_α, and the sign and magnitude of the coefficient indicate the correlation between f and χ_α. Large positive and negative coefficients indicate significant positive and negative correlations, respectively, while small coefficients indicate little or no correlation.

The Fourier spectrum also shows how f would be expressed as a linear combination of the basis functions:

$$f(x) = \sum_{\alpha \in \{0,1\}^n} \hat{f}(\alpha)\chi_\alpha(x) \tag{4.3}$$

Equation 4.3 is the inverse transform, and it shows how any f can be recovered from its Fourier representation.

The other representations used in this paper are obtained by changing the Fourier basis function definition so that logical ANDs and ORs are computed instead of XORs.

Below are definitions of AND (ξ) and OR (ζ) basis functions:

$$\xi_\alpha(x) = \begin{cases} +1 : \text{if } \sum_i \alpha_i x_i < \sum_i \alpha_i \\ -1 : \text{if } \sum_i \alpha_i x_i = \sum_i \alpha_i \end{cases} \tag{4.4}$$

$$\zeta_\alpha(x) = \begin{cases} +1 : \text{if } \sum_i \alpha_i x_i = 0 \\ -1 : \text{if } \sum_i \alpha_i x_i > 0 \end{cases} \tag{4.5}$$

By replacing the Fourier basis functions in Equation 4.2 with either of these sets of basis functions, we obtain "correlation spectra" for the AND and OR bases. That is, the coefficients reveal the correlation between f and the AND or OR functions, just as the Fourier coefficients do for the XOR functions. However, unlike in the XOR case, the coefficients obtained from Equation 4.2 do not generally give the linear combination of AND or OR functions that represents f. There is another transform equation that gives the linear combination (but not the correlation); however, only the correlation spectrum will be of interest here.

4.2.2 Spectral Learning

Given a set X of $\langle x, f(x)\rangle$ examples, a spectral learning algorithm attempts to learn a spectral representation of f that approximates it well. Since the number of basis functions is exponential in the number of inputs to a function, a spectral learning algorithm will typically select a subset of basis functions to use in its model, implicitly assigning coefficients of 0 to the remaining basis functions. If A is the set of labels of basis functions included in the model, then a spectral learner's approximation of f is given by the following:

$$\tilde{f}(x) = \begin{cases} +1 : \text{if } \sum_{\alpha \in A} \hat{f}(\alpha)\phi_\alpha(x) \geq 0 \\ -1 : \text{if } \sum_{\alpha \in A} \hat{f}(\alpha)\phi_\alpha(x) < 0 \end{cases} \tag{4.6}$$

where ϕ_α is a general basis function reference that could be replaced by any of the basis functions defined in the previous section.

In most applications of spectral learning, the basis functions included in the model are those whose coefficients appear to be largest, based on the training data, with coefficients estimated from the training data by the following:

$$\tilde{\hat{f}}(\alpha) = \frac{1}{|X|} \sum_{\langle x, f(x) \rangle \in X} f(x)\phi_\alpha(x) \tag{4.7}$$

Since this approach selects the basis functions that are most correlated with X, it will be referred to as the N-Most-Correlated approach. In this paper, after basis functions are selected, the coefficients are adjusted from the values given by Equation 4.7 to values that minimize squared error over the training data. This deviation from the standard approach usually gives better results [Drake and Ventura, 2011a].

A less common approach to spectral learning is to use a boosting algorithm to select basis functions [Jackson, 1997]. Figure 4.1 describes a spectral learning method based on the AdaBoost.M1 algorithm [Freund and Schapire, 1996]. In this approach, basis functions are selected iteratively based on correlation with a weighted data set, X_t (lines 5-6). Initially, each example is given equal weight (lines 2-3). However, after each basis function is selected, examples that are classified correctly are given less weight (lines 9-10). This causes subsequently selected basis functions to be increasingly well correlated with previously misclassified examples. (Note: For simplicity, Figure 4.1 assumes each ϕ_{α_t} is positively correlated with X_t. However, if ϕ_{α_t} is negatively correlated (i.e., if $\sum_{\langle x, f(x) \rangle \in X_t} f(x)\phi_{\alpha_t}(x) < 0$), then each occurrence of $\phi_{\alpha_t}(x)$ in lines 8 and 9 should be replaced with $(-\phi_{\alpha_t}(x))$.) The coefficients of selected basis functions are set by the boosting algorithm to the following:

$$\hat{f}(\alpha_t) = \pm \log\left(\frac{1-\epsilon_t}{\epsilon_t}\right) \tag{4.8}$$

BoostingBasisFunctionSelection(X, T)

(1) $A \leftarrow \emptyset$
(2) for each $\langle x, f(x)\rangle \in X$
(3) $w_{\langle x,f(x)\rangle} \leftarrow \frac{1}{|X|}$
(4) for $t = 1$ to T
(5) $X_t \leftarrow \{\langle x, w_{\langle x,f(x)\rangle} f(x)\rangle : \langle x, f(x)\rangle \in X\}$
(6) $\phi_{\alpha_t} \leftarrow$ SelectCorrelatedFunction(X_t)
(7) $A \leftarrow A \cup \phi_{\alpha_t}$
(8) $\epsilon_t \leftarrow \sum_{\{\langle x,f(x)\rangle:\phi_{\alpha_t}(x)\neq f(x)\}} w_{\langle x,f(x)\rangle}$
(9) for each $\langle x, f(x)\rangle \in X$ s.t. $\phi_{\alpha_t}(x) = f(x)$
(10) $w_{\langle x,f(x)\rangle} \leftarrow w_{\langle x,f(x)\rangle}(\frac{\epsilon_t}{1-\epsilon_t})$
(11) $z \leftarrow \sum_{\langle x,f(x)\rangle\in X} w_{\langle x,f(x)\rangle}$
(12) for each $\langle x, f(x)\rangle \in X$
(13) $w_{\langle x,f(x)\rangle} \leftarrow w_{\langle x,f(x)\rangle}/z$
(14) return A

Figure 4.1: The Boosting basis function selection procedure.

in which ϵ_t is the (weighted) misclassification rate of ϕ_{α_t} (as defined in Figure 4.1) and the sign of $\hat{f}(\alpha_t)$ is negative iff ϕ_{α_t} is negatively correlated with X_t. In the following sections, this spectral learning method is referred to as the Boosting approach.

4.3 Motivation

There are at least two reasons why a spectral learner might benefit from a multi-spectrum approach. One is that no representation will be best for all problems. This can be seen in Table 4.1, which compares the classification accuracy of single-spectrum learners using the AND, OR, and XOR bases with the N-Most-Correlated learning approach. In the table, the numbers in parentheses indicate the rank of each learner in highest accuracy order. Ties indicate no statistically significant difference between two results. (More precise details of the experiment are given later.) For each problem, one basis was significantly better than both of the others. In addition, each basis was significantly worse than both of the others at least once. Ideally, a spectral learner could choose an appropriate representation for each problem.

Table 4.1: Comparison of single-spectrum learners. (The number in parentheses is the rank of each learner in highest-accuracy order, with ties indicating no significant difference.) No representation is always best, and each is sometimes worst.

	AND	OR	XOR
Chess	81.1% (2)	89.1% (1)	75.4% (3)
German	71.0% (2)	70.6% (2)	72.0% (1)
Heart	81.9% (2)	80.1% (3)	83.8% (1)
Pima	73.9% (1)	73.1% (2)	73.1% (2)
SPECT	77.0% (3)	85.3% (1)	79.9% (2)
Voting	95.7% (1)	94.9% (2)	94.9% (2)
Wisc1	96.3% (1)	95.5% (2)	95.9% (2)
Wisc2	74.5% (1)	73.2% (2)	71.2% (3)
Wisc3	92.8% (2)	93.5% (1)	92.6% (2)

Another motivation is that a learner may be able to build a multi-spectrum model that is more accurate than any single-spectrum model it could construct. A single-spectrum learner is limited to the information and representational power of a single spectrum, while a multi-spectrum learner can combine spectra.

4.4 Multi-Spectrum Learning Methods

The following sections describe three learners: a Best-Basis learner, an Ensemble learner, and a Bag-Of-Features learner. Each uses multiple spectra in a different way.

4.4.1 Best-Basis

The Best-Basis learner attempts to determine which basis is best for a problem and then learns a model in that basis. In this paper, the bases considered by the learner are the AND, OR, and XOR bases, and the "best" basis is the one that maximizes estimated classification accuracy. Classification accuracy is estimated by randomly splitting the training data into training and validation sets (with 10% held out for validation) and computing the classification accuracy on the validation data when training on the training data. This process is repeated for 10 random splits of the data, and accuracies are averaged over those splits. The

basis that gives the highest average accuracy is then selected and a model is built in that basis using all of the training data.

While a single-spectrum learner's performance will sometimes be better than other single-spectrum learners and sometimes be worse (depending on the effectiveness of its basis for each problem), the Best-Basis learner can potentially do as well as the best single-spectrum learner on each problem. However, since the Best-Basis learner can only guess at which basis is best (by estimating test performance), it will sometimes choose a sub-optimal basis.

4.4.2 Ensemble

Although the Best-Basis learner has access to multiple bases, it uses a single basis in its final model. The Ensemble and Bag-Of-Features learners, on the other hand, combine spectra in their models.

The Ensemble learner uses multiple spectra by building an ensemble of single-spectrum models. Specifically, it builds a model in each of a set of bases (independently from one another) and then makes classifications by a majority vote of the single-spectrum models. The Ensemble learner's "meta"-model is defined by the following, in which $\tilde{f}_b$ is the model built in basis b (see Equation 4.6) and B is the set of bases used by the learner:

$$\tilde{f}_{ens}(x) = \begin{cases} +1 \ : \text{ if } \ \sum_{b \in B} \tilde{f}_b(x) \geq 0 \\ -1 \ : \text{ if } \ \sum_{b \in B} \tilde{f}_b(x) < 0 \end{cases}$$

In this paper, B contains the AND, OR, and XOR bases, so the Ensemble learner's classifications are the majority vote of models built in those three bases.

Intuitively, as long as each spectral model makes correct classifications most of the time, and as long as the individual models are not excessively redundant, it seems reasonable to expect that majority vote of the ensemble will be at least as accurate as any constituent model, if not more accurate. In spite of this, the Ensemble learner's model is not certain

to be more accurate than the best single-spectrum model. Specifically, it is possible for less accurate models in the ensemble to out-vote more accurate models, especially if the less-accurate models have highly correlated error.

4.4.3 Bag-Of-Features

The Bag-Of-Features learner is inspired by a feature selection perspective on spectral learning in which the basis functions are viewed as features and each basis is viewed as a bag of features. In principle, the Bag-Of-Features learner attempts to select the best features from each basis.

The Bag-Of-Features learner is like a single-spectrum learner except that it considers basis functions in all bases when selecting basis functions. Specifically, when selecting basis functions, whether using the N-Most-Correlated or Boosting approach, the learner searches through each of its candidate bases and selects the most correlated basis function(s) found. The Bag-Of-Features learner can be thought of as doing spectral learning with a hybrid "basis" made up of functions from each of the individual bases.

The Bag-Of-Features learner is the most powerful of the multi-spectrum learners, especially when using boosting, as it can directly combine basis functions from each basis to fit the data. (The Ensemble learner on the other hand selects basis functions in each basis separately and gives equal weight to each basis.) When using N-Most-Correlated learning, the Bag-Of-Features learner is somewhat more constrained, as it must select all basis functions based on correlation with the original unweighted data.

4.5 Results

The following sections analyze the performance of the multi-spectrum learners. The algorithms are compared on nine Boolean classification problems [Newman et al., 1998]. Some of the problems have one or more non-Boolean inputs, which were converted into Boolean inputs. Real-valued inputs were converted to Boolean inputs by applying a threshold (i.e.,

values greater than t were set to true). Each nominal input was converted into k Boolean inputs (one for each possible nominal value), with only the input corresponding to the correct value set to true.

The performance of each learner was tested by estimating classification accuracy on held-out data. The learners were tested on 100 random splits of the data with 10% held-out for testing each time. The results presented below are averages over those 100 trials. Statistical significance was measured pair-wise by a paired permutation test, with significant differences defined as those for which $p \leq 0.05$.

The number of basis functions used by each learner was set during training by estimating test accuracy for each number of basis functions from 1 to 128. Test accuracy was estimated by holding out 10% of the training data for validation and training on the remaining 90%. This process was repeated for ten random splits of the data, and then the learner was re-trained on all of the data with the number of basis functions that gave the highest average validation accuracy.

4.5.1 Single-Spectrum vs. Best-Basis

Table 4.2 compares the performance of the Best-Basis multi-spectrum learner to the single-spectrum learners. For each problem, the test accuracy of each learner is presented when using the Most-Correlated and Boosting spectral learning approaches. The number in parentheses is the rank of the learner in highest-accuracy order, relative to the others. A tie in rank indicates that there is not a significant difference between two results. A $*$ indicates that there is not a well-defined ranking in terms of significant differences (e.g., $A > B > C$, and there is a significant difference between A and C, but B is not significantly different from either).

While the Best-Basis learner does not always select the best basis, its overall performance is statistically indistinguishable from the best single-spectrum learner in 10 of 18 cases (4 of 9 cases for the N-Most-Correlated learning approach, and 6 of 9 cases for the

Table 4.2: Comparison of the Best-Basis learner and the AND, OR, and XOR single-spectrum learners. (The number in parentheses is the rank of each learner in highest-accuracy order, with ties indicating no significant difference, and * indicating an undefined ranking.) Although the Best-Basis learner does not always select the best basis, it often performs as well as the best single-spectrum learner.

N-Most-Correlated

	AND	OR	XOR	Best-Basis
Chess	81.1% (3)	89.1% (1)	75.4% (4)	89.0% (1)
German	71.0% (2)	70.6% (2)	72.0% (1)	71.2% (2)
Heart	81.9% (2)	80.1% (4)	83.8% (1)	81.7% (2)
Pima	73.9% (1)	73.1% (3)	73.1% (3)	73.8% (1)
SPECT	77.0% (4)	85.3% (1)	79.9% (3)	84.6% (1)
Voting	95.7% (1)	94.9% (2)	94.9% (2)	95.2% (2)
Wisc1	96.3% (*)	95.5% (*)	95.9% (*)	96.0% (*)
Wisc2	74.5% (1)	73.2% (2)	71.2% (4)	73.0% (2)
Wisc3	92.8% (2)	93.5% (1)	92.6% (2)	93.0% (2)

Boosting

	AND	OR	XOR	Best-Basis
Chess	99.3% (1)	97.2% (3)	98.8% (2)	97.2% (3)
German	72.4% (1)	72.4% (1)	68.1% (4)	72.6% (1)
Heart	81.4% (1)	78.1% (3)	75.1% (4)	80.9% (1)
Pima	73.7% (*)	73.1% (*)	72.8% (*)	73.3% (*)
SPECT	84.3% (1)	84.9% (1)	71.5% (4)	84.7% (1)
Voting	95.3% (*)	94.9% (*)	94.7% (*)	95.1% (*)
Wisc1	96.4% (1)	96.2% (1)	95.2% (4)	95.9% (3)
Wisc2	71.2% (1)	69.7% (1)	64.4% (3)	69.7% (1)
Wisc3	93.9% (2)	94.5% (1)	92.6% (4)	93.7% (2)

Boosting approach, including the cases where the full ranking is undefined). In addition, the Best-Basis learner's performance ranks no worse than second in 16 of 18 cases (9 of 9 cases for N-Most-Correlated, and 7 of 9 for Boosting).

Table 4.3 shows how often the Best-Basis learner selects each basis. The basis (or bases, if there is a statistical tie) of the best single-spectrum learner is highlighted in bold. In most cases, the learner tends to select the best basis, and in several cases it does so consistently. Other times, however, the learner is less consistent. In these cases, a small performance difference between bases and/or a relatively small amount of training data makes consistently choosing the best basis difficult. Interestingly, in a few cases the learner

Table 4.3: Percentage of trials in which the Best-Basis learner selected each basis. (The basis of the best single-spectrum learner is highlighted in bold.) On many problems, the Best-Basis learner consistently selects the best basis. In some cases, however, it consistently selects a sub-optimal basis.

	N-Most-Correlated			Boosting		
	AND	OR	XOR	AND	OR	XOR
Chess	0%	**100%**	0%	**0%**	100%	0%
German	14%	13%	**73%**	**80%**	**20%**	0%
Heart	60%	7%	**33%**	**94%**	6%	0%
Pima	**63%**	35%	2%	**56%**	0%	44%
SPECT	0%	**92%**	8%	**4%**	**96%**	0%
Voting	**75%**	17%	8%	**85%**	7%	8%
Wisc1	**64%**	12%	24%	**53%**	**0%**	47%
Wisc2	**36%**	13%	51%	**50%**	**35%**	15%
Wisc3	74%	**11%**	15%	89%	**0%**	11%

consistently selects a sub-optimal basis, as the validation method consistently leads the learner to choose a basis that does not generalize as well.

4.5.2 Single-Spectrum vs. Ensemble

The Ensemble learner does a better job than the Best-Basis learner of matching the performance of the best single-spectrum learner. In Table 4.4, the Ensemble learner matches or exceeds the performance of the best single-spectrum learner in 15 of 18 cases (6 of 9 cases when using the N-Most-Correlated approach, and 9 of 9 cases when using the Boosting approach). (The Ensemble learner's performance is indistinguishable from the best single-spectrum learner's performance in those cases where a full ranking is undefined.) In addition, the Ensemble learner's performance is never worse than second-best.

Unfortunately, however, although the Ensemble learner is effective at matching the performance of the best single-spectrum learner, consistently outperforming the best single-spectrum learner seems difficult. In Table 4.4, only once (when using Boosting on the *Voting* data set) is the Ensemble learner's performance significantly better than the best single-spectrum learner.

Table 4.4: Comparison of the Ensemble learner and the AND, OR, and XOR single-spectrum learners. (The number in parentheses is the rank of each learner in highest-accuracy order, with ties indicating no significant difference, and * indicating an undefined ranking.) The Ensemble learner usually matches or exceeds the performance of the best single-spectrum learner.

N-Most-Correlated

	AND	OR	XOR	Ensemble
Chess	81.1% (3)	89.1% (1)	75.4% (4)	88.3% (2)
German	71.0% (3)	70.6% (3)	72.0% (1)	71.6% (1)
Heart	81.9% (3)	80.1% (4)	83.8% (1)	83.9% (1)
Pima	73.9% (1)	73.1% (3)	73.1% (3)	73.9% (1)
SPECT	77.0% (4)	85.3% (1)	79.9% (3)	83.3% (2)
Voting	95.7% (1)	94.9% (2)	94.9% (2)	95.5% (1)
Wisc1	96.3% (1)	95.5% (3)	95.9% (3)	96.3% (1)
Wisc2	74.5% (1)	73.2% (2)	71.2% (4)	73.0% (2)
Wisc3	92.8% (3)	93.5% (1)	92.6% (3)	93.6% (1)

Boosting

	AND	OR	XOR	Ensemble
Chess	99.3% (1)	97.2% (4)	98.8% (3)	99.3% (1)
German	72.4% (1)	72.4% (1)	68.1% (4)	73.1% (1)
Heart	81.4% (1)	78.1% (3)	75.1% (4)	81.1% (1)
Pima	73.7% (*)	73.1% (*)	72.8% (*)	73.5% (*)
SPECT	84.3% (1)	84.9% (1)	71.5% (4)	84.9% (1)
Voting	95.3% (2)	94.9% (3)	94.7% (3)	95.5% (1)
Wisc1	96.4% (1)	96.2% (1)	95.2% (4)	96.4% (1)
Wisc2	71.2% (*)	69.7% (*)	64.4% (*)	72.3% (*)
Wisc3	93.9% (3)	94.5% (1)	92.6% (4)	94.8% (1)

4.5.3 Single-Spectrum vs. Bag-Of-Features

Although the Bag-Of-Features learner is the most powerful of the multi-spectrum learners (especially when using boosting), its overall performance is the worst. In Table 4.5, the Bag-Of-Features learner matches the performance of the best single-spectrum learner in only 5 of 18 cases (4 of 9 cases when using the N-Most-Correlated approach, and 1 of 9 cases when using the Boosting approach), and it is outperformed by two single-spectrum learners four times. (When doing N-Most-Correlated learning, the Bag-Of-Features learner's performance is indistinguishable from the performance of the best single-spectrum learner on the *Pima* problem, but not on the *Wisc2* problem.)

Table 4.5: Comparison of the Bag-Of-Features learner and the AND, OR, and XOR single-spectrum learners. (The number in parentheses is the rank of each learner in highest-accuracy order, with ties indicating no significant difference, and * indicating an undefined ranking.) The Bag-Of-Features learner is usually outperformed by one or more single-spectrum learners.

N-Most-Correlated

	AND	OR	XOR	Bag
Chess	81.1% (2)	89.1% (1)	75.4% (4)	81.1% (2)
German	71.0% (2)	70.6% (2)	72.0% (1)	71.0% (2)
Heart	81.9% (2)	80.1% (3)	83.8% (1)	79.9% (3)
Pima	73.9% (*)	73.1% (*)	73.1% (*)	73.5% (*)
SPECT	77.0% (4)	85.3% (1)	79.9% (3)	84.9% (1)
Voting	95.7% (1)	94.9% (2)	94.9% (2)	95.7% (1)
Wisc1	96.3% (1)	95.5% (2)	95.9% (2)	95.6% (2)
Wisc2	74.5% (1)	73.2% (*)	71.2% (*)	72.2% (*)
Wisc3	92.8% (3)	93.5% (1)	92.6% (3)	93.4% (1)

Boosting

	AND	OR	XOR	Bag
Chess	99.3% (1)	97.2% (4)	98.8% (2)	98.8% (2)
German	72.4% (1)	72.4% (1)	68.1% (3)	68.8% (3)
Heart	81.4% (1)	78.1% (2)	75.1% (4)	77.1% (2)
Pima	73.7% (1)	73.1% (2)	72.8% (2)	73.0% (2)
SPECT	84.3% (1)	84.9% (1)	71.5% (4)	81.4% (3)
Voting	95.3% (1)	94.9% (2)	94.7% (2)	94.6% (2)
Wisc1	96.4% (1)	96.2% (1)	95.2% (4)	96.1% (1)
Wisc2	71.2% (1)	69.7% (1)	64.4% (4)	67.0% (3)
Wisc3	93.9% (2)	94.5% (1)	92.6% (4)	93.8% (2)

When using the N-Most-Correlated learning approach, one problem with the Bag-Of-Features learner is that it often selects all basis functions from a single basis. (This occurs whenever the N most correlated basis functions are all in one basis.) Table 4.6 shows how often this occurred over the 100 random splits of the data into training and test sets. It also shows the average percentage of basis functions selected from each basis (not counting the 0^{th}- and 1^{st}-order basis functions, which are the same in each basis). The Bag-Of-Features learner frequently learns a single-spectrum model on the *Chess*, *SPECT*, *Wisc1*, *Wisc2*, and *Wisc3* problems, and to a lesser extent on the *German* problem. In the case of the *SPECT* and *Wisc3* problems, the basis functions were selected from the best basis for the problem,

Table 4.6: Average percentage of functions from each basis in the models learned by the Bag-Of-Features learner when doing N-Most-Correlated learning, and the percentage of trials in which functions were selected from one basis only. The learner often ends up with a model that uses only a single basis, and for some problems (*Chess*, *German*, *Wisc1*, *Wisc2*) the basis it favors is not the best choice.

	Model Composition			Single-Basis
	AND	OR	XOR	Model %
Chess	100.0%	0.0%	0.0%	100%
German	0.3%	84.2%	15.5%	22%
Heart	16.8%	77.1%	6.1%	1%
Pima	1.4%	74.0%	24.6%	4%
SPECT	0.0%	100.0%	0.0%	100%
Voting	53.4%	44.3%	2.3%	0%
Wisc1	1.7%	98.3%	0%	79%
Wisc2	10.3%	0%	89.7%	77%
Wisc3	1.0%	98.6%	0.3%	93%

allowing the Bag-Of-Features learner to match the performance of the best single-spectrum learner, even if it could not exceed it. For the *Chess*, *German*, *Wisc1*, and *Wisc2* problems, however, it selected basis functions from a sub-optimal basis.

It is interesting to compare the Bag-Of-Features learner's single-spectrum models with those of the Best-Basis learner. In those cases where the Bag-Of-Features learner ends up with a single-spectrum model, it implicitly selects the basis that has the N basis functions that are most correlated (individually) with the training data. This will typically be the basis that best fits the training data. The Best-Basis learner, on the other hand, attempts to select the basis that will perform best on unseen data by choosing the basis that maximizes accuracy on held-out data. On problems for which the Bag-Of-Features learner often learns a single-spectrum model, the Best-Basis approach seems slightly better. The learners tend to choose different bases on four of those problems (*Chess*, *German*, *Wisc1*, *Wisc3*), and the Best-Basis learner favors the better basis on all but one of those (*Wisc3*).

When using Boosting, the Bag-Of-Features learner rarely selects all functions from a single basis. However, it does tend to select most functions (approx. 79% on average) from the XOR basis. A likely reason for this is that the XOR basis functions are all orthogonal

to one another. Since AND and OR functions are somewhat correlated with each other, it is probably easier to repeatedly find XOR basis functions that provide "new information." Unfortunately, this information is with respect to the training sample only. Table 4.5 shows that models built when using boosting in the XOR basis did not generalize as well, but this is not considered in the Bag-Of-Features/Boosting approach. In contrast, the Best-Basis learner can recognize this and usually avoided the XOR basis when using boosting (see Table 4.3).

4.5.4 Multi-Spectrum Comparison

In a direct comparison of multi-spectrum learners (Table 4.7), the Ensemble learner outperforms the other multi-spectrum learners. It matches or exceeds the performance of the other learners in 16 of 18 cases. (The other cases are *Chess* and *SPECT* with N-Most-Correlated learning. For *SPECT*, the full ranking is not defined, but the Ensemble learner did significantly worse than the Bag-Of-Features learner.)

By combining the models built in each basis, the Ensemble learner has access to more information than the Best-Basis learner when making classifications. However, since the models in each basis are learned independently, and since classifications are "averaged" over these models, the Ensemble learner is less susceptible to overfitting than the Bag-Of-Features learner.

4.6 Conclusion

Spectral learning algorithms learn an unknown function by learning a spectral representation of the function; however, no representation will be best for all problems. This paper has presented and compared methods for learning from multiple spectra. The best multi-spectrum approach appears to be an ensemble approach in which classifications are made by a majority vote over a set of models built independently in each of a set of candidate representations. The ensemble approach is better at matching the performance of the best

Table 4.7: Comparison of multi-spectrum learners. (The number in parentheses is the rank of each learner in highest-accuracy order, with ties indicating no significant difference, and * indicating an undefined ranking.) The Ensemble learner usually matches or exceeds the performance of the other multi-spectrum learners.

N-Most-Correlated

	Best-Basis	Ensemble	Bag
Chess	89.0% (1)	88.3% (2)	81.1% (3)
German	71.2% (1)	71.6% (1)	71.0% (1)
Heart	81.7% (2)	83.9% (1)	79.9% (3)
Pima	73.8% (1)	73.9% (1)	73.5% (1)
SPECT	84.6% (*)	83.3% (*)	84.9% (*)
Voting	95.2% (*)	95.5% (*)	95.7% (*)
Wisc1	96.0% (1)	96.3% (1)	95.6% (3)
Wisc2	73.0% (1)	73.0% (1)	72.2% (1)
Wisc3	93.0% (*)	93.6% (*)	93.4% (*)

Boosting

	Best-Basis	Ensemble	Bag
Chess	97.2% (3)	99.3% (1)	98.8% (2)
German	72.6% (1)	73.1% (1)	68.8% (3)
Heart	80.9% (1)	81.1% (1)	77.1% (3)
Pima	73.3% (1)	73.5% (1)	73.0% (1)
SPECT	84.7% (1)	84.9% (1)	81.4% (3)
Voting	95.1% (2)	95.5% (1)	94.6% (2)
Wisc1	95.9% (*)	96.4% (*)	96.1% (*)
Wisc2	69.7% (2)	72.3% (1)	67.0% (3)
Wisc3	93.7% (2)	94.8% (1)	93.8% (2)

single representation than an approach that attempts to select the best single representation, and it is less prone to overfitting than an approach that attempts to combine the best basis functions of each representation into a single model.

One interesting question for future work is the question of which spectral representations a multi-spectrum learner should consider. For multi-spectrum learning approaches that combine representations, there is the additional question of which representations are useful in combination. For example, is it bad to have representations that are too "similar" to each other? How "different" should the representations be, and how many should be

considered? Finally, a potentially interesting direction for future work is to determine if a learner can effectively generate bases on-the-fly to adapt to specific learning problems.

Chapter 5

Sentiment Regression: Using Real-Valued Scores to Summarize Overall Document Sentiment

Abstract

In this paper, we consider a sentiment regression problem: summarizing the overall sentiment of a review with a real-valued score. Empirical results on a set of labeled reviews show that real-valued sentiment modeling is feasible, as several algorithms improve upon baseline performance. We also analyze performance as the granularity of the classification problem moves from two-class (positive vs. negative) towards infinite-class (real-valued).

5.1 Introduction

Sentiment classification is the problem of classifying the opinion or feeling of written text. It has many potential applications including systems for automatic product recommendation, "flame" detection in online forums, assigning ratings to written reviews, organizing written surveys by satisfaction level, email filtering, and organizing/summarizing reviews of products by feature.

Previous work in sentiment analysis has considered classification scenarios involving just a few sentiment categories. In some applications, this coarse-grained view of sentiment may be sufficient. In other situations, however, such a coarse-grained analysis may be unacceptable. Furthermore, even in cases where a coarse-grained analysis is acceptable, more fine-grained sentiment distinctions, if possible, would generally be preferred.

In this paper, we examine the ability of learning algorithms to make precise, fine-grained assessments of overall sentiment. Specifically, we consider the problem of summarizing the overall sentiment of a review by labeling it with a real-valued score. In doing so, we introduce a real-world data set of reviews labeled with scores on a 91 point scale (1.0 to 10.0 in increments of 0.1) and compare the performance of several machine learning algorithms on the task of predicting the score given by the author.

In addition, we compare the performance of the algorithms as the granularity of the sentiment categorization moves from two-class (positive vs. negative) towards infinite-class (real-valued). As the granularity becomes increasingly fine, it seems reasonable to expect that a regression approach to sentiment analysis will eventually be more appropriate than a classification approach. Consequently, we compare the performance of both regression- and classification-based algorithms as the number of sentiment categories increases.

5.2 Related Work

Early work in classifying the overall sentiment of reviews as positive or negative includes the work of Turney [Turney, 2002] and Pang, Lee, and Vaithyanathan [Pang et al., 2002]. Turney used the mutual information between phrases and specific semantic words to find positive and negative clauses, which were used in turn to determine overall document sentiment, while Pang, Lee, and Vaithyanathan applied machine learning algorithms to the task. Pang and Lee [Pang and Lee, 2005] later extended their work to more fine-grained 3-class (positive, negative, or neutral) and 4-class (x out of 4 stars) scenarios, and they introduced a method for allowing standard classification algorithms to make use of the natural ordering of sentiment classes. Bikel and Sorensen [Bikel and Sorensen, 2007] presented results of applying an averaged perceptron with a word subsequence kernel to user reviews from Amazon.com (on a 5 point scale), finding that it can accurately distinguish between reviews that are above and below a chosen score.

In other related work, Yu and Hatzivassiloglou [Yu and Hatzivassiloglou, 2003] have presented work on distinguishing between opinions and facts at both the sentence and document level. Dave, Lawrence, and Pennock [Dave et al., 2003] and Hu and Liu [Hu and Liu, 2004] have presented results of work on extracting feature-specific sentiments from reviews of a single product and presenting summaries (by feature) of the results. Meanwhile, Wilson, Wiebe, and Hwa [Wilson et al., 2004] presented experimental results on a problem related to sentiment classification: determining the strength of an opinion.

5.3 Real-Valued Sentiment Analysis

Performing sentiment analysis on a real-valued scale can be viewed as a generalization of the typical sentiment classification scenario. Many of the issues in sentiment classification, such as differences in word usage and meaning between authors, become even more pronounced as the desired precision increases. For example, consider the difficulty of accurately inferring

a real-valued score from a review when one author might describe a product given a score of 80 out of 100 as "excellent" while a more demanding author may reserve that term for products with scores above 95.

To test the ability of learning algorithms to make fine-grained sentiment distinctions, a set of video game reviews was collected from GameSpot.com. Each game review is labeled with a score between 1.0 and 10.0, rounded to one decimal point. (Since scores are rounded to one decimal point there is a finite set of 91 possible scores; nevertheless, a real-valued perspective seems appropriate.)

All reviews from the months of June 2005 through December 2006 were collected, resulting in a total of 1,822 reviews (roughly 96 per month). The average review length is approximately 1,300 words, but the length varies significantly, with the smallest review containing only 93 words and the largest containing 4,638. There are 31 different authors represented in the data, although some contributed more reviews than others (several authors contributed only a few reviews, while just over half of the reviews came from the seven most common authors).

5.4 Feature Selection

In the experiments described in the following sections, there were typically around 30,000 unique words in the training data. However, the learning algorithms generally performed better when the vocabulary was reduced to a smaller subset, V, of those words.

In order to determine which words should be considered, we used a word-score correlation metric. The correlation of a word w with the scores of a set of reviews R, denoted $c(w, R)$, was defined by the following:

$$c(w,R) = \frac{1}{|R|} \sum_{r \in R} \left\{ I(w,r) \cdot \left(S(r) - \frac{1}{|R|} \sum_{r' \in R} S(r') \right) \right\}$$

where $S(r)$ is the real-valued score associated with review r and $I(w, r)$ is a function that outputs 1 if r contains word w and outputs -1 otherwise.

Note that the $\frac{1}{|R|}\sum_{r' \in R} S(r')$ term is the average review score. Intuitively, if a word is positively correlated with review scores then it would tend to appear in documents with above average scores and be absent from reviews with below average scores. Similarly, if a word is negatively correlated with review scores then it would tend to appear in documents with below average scores and be absent from reviews with above average scores.

To see how this applies in the correlation metric defined above, notice that if a word w appears in a review r and r's score is above average, then both $I(w, r)$ and $(S(r) - \frac{1}{|R|}\sum_{r' \in R} S(r'))$ are positive, and the correlation goes up. If w does not appear in review r and r's score is below average, then both terms are negative, and again the correlation goes up. Meanwhile, in the other two cases (when w is not in r and r's score is above average and when w is in r and r's score is below average), the terms have different signs and the correlation drops.

This metric reveals how much a word's presence/absence tends to cause a review's score to deviate from the mean on average. A large positive value indicates that the word tends to occur in reviews with above average scores and be absent from reviews with below average scores, while a large negative value indicates the opposite. A value near 0 indicates that the word's presence does not tend to influence the score significantly in either a positive or negative direction. This metric implicitly tends to remove words that occur too rarely or too frequently to be useful for learning.

Table 5.1 shows the top 10 positively and negatively correlated words over the entire set of reviews from June 2005 through December 2006.

It is interesting to note in Table 5.1 that some of the most correlated features, particularly on the positive side, do not carry obviously positive sentiment. However, they appear to be used so much more often in phrases of positive sentiment that there is a significant positive correlation between their usage and the score of the review.

Table 5.1: The top 10 positively and negatively correlated words, according to the word-score correlation metric.

Positive		Negative	
0.514	great	-0.251	dull
0.369	quite	-0.236	generic
0.353	excellent	-0.199	decent
0.309	new	-0.196	repetitive
0.301	experience	-0.195	ugly
0.300	definitely	-0.187	boring
0.291	best	-0.183	bland
0.290	expect	-0.170	poor
0.290	year	-0.168	terrible
0.285	unique	-0.166	poorly

Since both positive and negative correlations are useful for learning, the words added to the vocabulary were those that maximized the absolute value of $c(w, R)$. This has the effect of selecting the words whose presence/absence causes the score to deviate from the mean the most on average.

The number of vocabulary words used by each learning algorithm is a parameter that is tuned by validation along with any other algorithm-specific parameters. (The validation process is described later in the Results section.) Using the set V of vocabulary words, each review r is converted into a Boolean vector $\vec{x} = \{x_1, x_2, ..., x_{|V|}\}$ in which x_i is true if and only if the i^{th} vocabulary word appears in r. All of the learning algorithms presented in the following section used these Boolean vectors as input features. Thus, the algorithms learn to predict the score of a review based solely on presence/absence of words in the review.

5.5 Learning Algorithms

Four learning methods, two classification-based and two regression-based, were used in the sentiment experiments: a Naive Bayes classification algorithm, a linear regression algorithm, and classification and regression support vector machine (SVM) algorithms.

5.5.1 Naive Bayes

The Naive Bayes algorithm treats each possible score as a different class. Given a review to classify, it will predict the class c that is most likely given the feature vector $\vec{x}$, under the assumption that the input features are conditionally independent of each other given c:

$$\begin{aligned}\operatorname*{argmax}_c P(c|\vec{x}) &= \operatorname*{argmax}_c \frac{P(c)P(\vec{x}|c)}{P(\vec{x})} \\ &= \operatorname*{argmax}_c P(c)P(\vec{x}|c) \\ &= \operatorname*{argmax}_c P(c)\prod_i P(x_i|c)\end{aligned}$$

The probabilities $P(c)$ and $P(x_i|c)$ are estimated from counts in the training data. In order to smooth the probability distribution, the counts used to estimate $P(x_i|c)$ in our experiments were incremented by a small value δ, which was set to the value that gave the best result in the validation process described in the next section.

5.5.2 Linear Regression

The linear regression algorithm attempts to learn a function f that maps input vectors to scores. It represents f by a linear combination of the input features:

$$f(\vec{x}) = w_0 + \sum_i w_i x_i$$

We used a forward step-wise approach to set the weights and perform feature selection. Initially, w_0 was set to the mean review score. Then, until the desired number of input features had been added, input features were added incrementally. At each step, the feature that would reduce squared error most if added was selected. When added, its weight was set such that squared error would be minimized, given the features and weights already added. The number of input features to include was determined by validation. (Note that since the

linear regression algorithm performed its own feature selection, it did not use the word-score correlation metric to choose vocabulary words.)

5.5.3 SVM

The two SVM algorithms are based on the classification and regression variants of Joachim's SVM^{light} [Joachims, 1999]. In the experiments reported below, the default settings of SVM^{light} were used.

The regression SVM, like linear regression, learned a function that mapped the Boolean feature vectors to scores. The classification SVM, on the other hand, like Naive Bayes, treated each possible score as a unique class. In order to use the SVM in a multi-class scenario, n binary classifiers were created, each used to distinguish one of the n classes from the others. Test instances were classified by choosing the classifier that reported the largest positive distance from its decision surface.

5.6 Results

The experiments on the GameSpot data were conducted as follows. The reviews of the six months from July 2006 through December 2006 were used as test data, with each month tested separately. When testing on a particular month, the algorithms were given the previous 12 months of reviews as training data. This resulted in training sets of roughly 1,000 labeled reviews.

The parameters of the algorithms, including the number of vocabulary words to use, were set by using the previous month as a validation set. Specifically, the parameter settings that gave the best results in month i (with months $i-12$ through $i-1$ used as training data) were used as the parameter settings when testing on month $i+1$.

Classification accuracy, the percentage of correctly classified examples, is an effective performance metric in a 2-class sentiment classification scenario. However, as the sentiment spectrum is subdivided into more categories, classification accuracy becomes less meaningful

Table 5.2: Average squared error on the real-valued sentiment prediction task. All algorithms outperform the baseline on average, while the SVM Regression algorithm has the lowest error overall.

Test Set	# Reviews	Baseline	Linear Regr	Naive Bayes	SVM-Class	SVM-Regr
Jul 2006	53	3.07	1.49	3.19	3.31	1.33
Aug 2006	50	2.60	1.10	1.69	1.83	0.87
Sept 2006	89	1.63	0.89	1.49	2.18	0.84
Oct 2006	118	2.29	1.45	1.75	1.66	1.45
Nov 2006	157	2.50	1.11	1.80	1.81	0.99
Dec 2006	124	2.32	1.40	1.93	1.75	0.92
Average		2.35	1.24	1.89	1.96	1.06

(and less realistic). It becomes more important to predict a sentiment that is "close" to the true sentiment. Thus, the performance metric we use to compare the algorithms is mean squared error, or the average squared distance between the score given by an author and the score predicted by a learning algorithm:

$$\frac{1}{R}\sum_{r\in R}(score(r) - prediction(r))^2$$

This metric requires that the sentiment categories be mapped to real numbers. In our case, the reviews are already labeled with scores, although one could imagine any sentiment categorization being mapped to reasonable real values. Note that when there are two classes/values, there is a direct relationship between squared error and classification accuracy, as lower squared error always implies higher classification accuracy, and vice versa.

To provide a point of reference on the effectiveness of the algorithms, a simple baseline algorithm was also tested. The baseline algorithm uses the mean score of the training data as its prediction on future data.

5.6.1 Real-Valued Sentiment Prediction

Table 5.2 shows the average squared error of each of the algorithms on the six months of test data. The average error is a weighted average, with each month weighted by the number of

examples that month (i.e., it is the per-example average over the six month time period.) The classification algorithms (Naive Bayes and SVM Classification) treat each possible score as a unique class. Since the review scores are rounded to one decimal point in the 1.0-10.0 range, there are potentially 91 classes; however, the algorithms only consider classes observed during training, and many of the possible scores do not occur in every training set. (On average, there were 75 unique scores per training set.) The regression algorithms (Linear Regression and SVM Regression) treat the scores as real-valued outputs of an unknown function. When testing, the real-valued predictions of the regression algorithms are rounded to the nearest valid score.

Table 5.2 reveals that each of the algorithms outperformed the simple baseline. The best performing algorithm was the regression-based SVM, which had an average squared error of 1.06, compared to the baseline squared error of 2.35. In terms of the absolute error, or the average distance between the true and predicted score, the SVM Regression and baseline algorithms erred by 0.76 and 1.21 points, respectively, on average. (If the distribution of scores were uniformly spread across the 1.0 to 10.0 range, we would expect the baseline algorithm to be off by about 2.25 on average. However, the distribution of scores for these reviews is somewhat concentrated around a mean value of about 7, so baseline performance is better.) Thus, while non-uniformity in the distribution allows the baseline algorithm's predictions to be off by only 1.21 points on the 10 point scale, the SVM algorithm's predictions were nearly half a point closer on average.

Given that the algorithms are making predictions on the basis of word presence alone, this result is encouraging, and it suggests that learning to make accurate predictions on a fine-grained sentiment scale is feasible. We suspect that there is still room for improvement on this sentiment analysis task, and we expect that the application and development of more sophisticated techniques will lead to improved results.

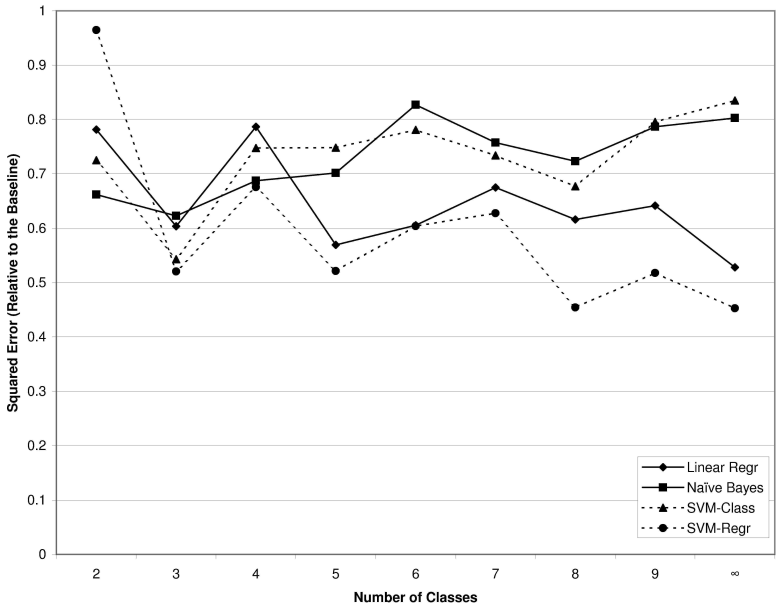
5.6.2 Classification vs. Regression

Of the four algorithms tested on the GameSpot reviews, the classification algorithms performed the worst. This is not surprising, given that the regression algorithms naturally incorporate the concept that nearby sentiment values are close, while to the classification algorithms there is no similar concept of closeness between classes (although a method for explicitly incorporating a measure of closeness between sentiment classes was successfully applied by Pang & Lee [Pang and Lee, 2005] in 3- and 4-class scenarios). Furthermore, the ability of classification algorithms to make precise sentiment distinctions is limited by the fact that their precision is limited to the number of classes they consider, and as the number of classes increases the number of examples of each class becomes small.

Figure 5.1 demonstrates the effectiveness of the algorithms as the granularity of the sentiment spectrum moves from two classes towards the full [1.0...10.0] range. In this experiment, the 10 point scale was subdivided into regions of equal size, and review scores were set to the midpoint of their region. Thus, for example, in the two class case, the [1.0...10.0] range was divided into the regions [1.0...5.5] and (5.5...10.0], with scores in each region set to 3.25 and 7.75, respectively. (Reviews with scores on the boundary between two regions were arbitrarily assigned to the lower region.) The figure shows algorithm performance relative to the baseline algorithm. Specifically, the figure plots each algorithm's squared error divided by the baseline squared error. The actual squared errors of the algorithms are shown in Table 5.3.

Interestingly, in the two-class case, the two best results came from the classification algorithms. In the three- and four-class scenarios, results were mixed. Beyond four classes, the regression algorithms were always superior. Also noteworthy is the fact that the SVM regression algorithm was the top performer in all experiments with more than two classes (although in the 6-class scenario it was matched by linear regression). These results suggest that unless the number of sentiment categories will be quite small, a regression approach will likely be best.

Figure 5.1: Average squared error, relative to the baseline, as the number of sentiment classes increases. Although the classification algorithms (Naive Bayes and SVM Classification) initially perform better than the regression algorithms (Linear and SVM Regression), the regression algorithms perform better as the number of sentiment classes grows.



5.7 Conclusion

In this paper, we have considered a real-valued approach to sentiment analysis, and compared the performance of several learning algorithms at the task of assigning a real-valued score to a review. Empirical results suggest that learning to accurately evaluate sentiment on such a fine-grained scale is possible. Using a simple approach based on the presence and absence of words, the SVM regression algorithm reduced the squared error of the baseline algorithm by more than half. In absolute terms, its predictions were off by an average of 0.76 points (compared to a baseline of 1.21 points) on the 1.0-10.0 scale.

We expect that this result can be improved as more sophisticated sentiment analysis techniques are applied. Possible areas for improvement include accounting for contextual changes in the sentiment of a word (e.g., "not good" vs. "good") [Wilson et al., 2005], identifying subjective portions of the reviews and applying the algorithm on just those portions

Table 5.3: Average squared error for increasingly fine-grained sentiment categorization. Although the classification algorithms (Naive Bayes and SVM Classification) initially perform better than the regression algorithms (Linear and SVM Regression), the regression algorithms perform better as the number of sentiment classes grows.

# Classes	Baseline	Linear Regr	Naive Bayes	SVM-Class	SVM-Regr
2	4.87	3.80	3.22	3.53	4.69
3	4.77	2.88	2.97	2.59	2.48
4	2.85	2.24	1.96	2.13	1.93
5	3.31	1.89	2.32	2.48	1.73
6	2.56	1.55	2.12	2.00	1.55
7	2.48	1.68	1.88	1.82	1.56
8	2.63	1.62	1.91	1.78	1.20
9	2.44	1.56	1.92	1.94	1.26
∞	2.35	1.24	1.89	1.96	1.06

[Pang and Lee, 2004], or identifying the reviewer's sentiment with respect to specific aspects of the product [Popescu and Etzioni, 2005].

As expected, the results also suggest that regression approaches will outperform classification approaches as the number of sentiment classes approaches a real-valued scale. In fact, our experiments showed that the regression and classification algorithms performed similarly when there were three or four classes, while beyond four classes the regression algorithms always performed better.

Chapter 6

Using Spectral Features to Improve Sentiment Analysis

Abstract

A common approach to sentiment classification is to identify a set of sentiment-carrying words and then to use machine learning to build a classifier that can classify sentiment based on the presence/absence of those words. In this paper, we propose a Fourier-based extension of this approach. Specifically, we introduce a spectral learning algorithm that implicitly identifies sentiment-carrying words and higher-order functions of those words as it learns to assign real-valued sentiment scores to documents. The spectral learner extends the word presence model by applying Boolean logic operators (AND, OR, and XOR) to the word presence features to identify useful higher-order features. These spectral features can be used in other learning algorithms, and we show how the performance of other learning algorithms can be improved by these features. Finally, we consider the problem of determining which of a pair of reviews expresses more positive overall sentiment, and we show that the spectral learner can identify very small distinctions in sentiment with better-than-random accuracy, while larger distinctions can be correctly identified with high accuracy.

6.1 Introduction

An important problem in sentiment analysis and opinion mining is sentiment classification: the problem of determining whether an opinion is expressing positive or negative sentiment. There are many practical uses for sentiment classification, including the automatic classification of written reviews [Pang et al., 2002], the automatic aggregation of opinions about product features [Dave et al., 2003, Liu et al., 2005, Nasukawa and Yi, 2003], the summarization of product reviews [Hu and Liu, 2004], the identification of important product features [Popescu and Etzioni, 2005], and the identification of strong opinions (e.g., angry, ranting forum posts) [Wilson et al., 2004].

Sentiment classification is generally used to make classifications on a fairly coarse scale. In previous work, the sentiment task has usually been to classify documents or sentences as either positive or negative [Pang et al., 2002], or as belonging to one of a few sentiment classes (e.g., x out of 4 stars [Pang and Lee, 2005]). In this paper, however, sentiment classification is viewed as a regression problem, and the sentiment task is to assign a real-valued score (on a 1.0-10.0 scale) to a product based on a written review.

In this paper, we introduce a feature selection method for sentiment analysis that is inspired by the Fourier-based learning algorithms developed in computational learning theory. These algorithms attempt to learn a function by learning a spectral representation of the function. Fourier-based algorithms have been successfully used to prove interesting learnability results about various classes of functions [Jackson et al., 2002, Kushilevitz and Mansour, 1993, Klivans et al., 2004]. Fourier-based algorithms have also been successfully applied in real-world domains [Kargupta and Park, 2004, Mansour and Sahar, 2000].

The spectral learning approach we present in this paper is a novel method designed specifically for high-dimensional natural language problems. It combines ideas from the low-order [Linial et al., 1993] and boosting [Jackson, 1997] Fourier-based algorithms of learning theory with the generalized Fourier-based learning [Drake and Ventura, 2005] and multi-

spectrum [Drake and Ventura, 2011b] approaches that have been effective in real-world settings.

In sentiment analysis applications, a standard approach is to identify a set of words that are predictive of sentiment and then to apply a machine learning algorithm to learn to classify sentiment based on the presence/absence of those words. Our spectral learning algorithm implicitly identifies useful sentiment words, and it enhances the standard word presence approach by identifying useful higher-order functions of the sentiment words. Specifically, the spectral approach applies Boolean operators (AND, OR, and XOR) to the word presence features to identify higher-order features that provide useful information that was not captured in the first-order word presence features.

Perhaps the most interesting observation of this paper is that these spectral features can be used in other learning algorithms to improve their performance. We show how the performance of linear regression and support vector machine (SVM) learners is significantly improved when the spectral method is used to provide sentiment features for these algorithms. This result provides an interesting contrast to previous unsuccessful extensions of the word presence model, such as word frequency and bi-grams, that have been shown not to result in better performance [Pang et al., 2002].

We conclude the paper by considering a related sentiment classification problem: determining which of two products should be assigned a higher overall rating based on the written reviews, and we show that even when the ratings differ by only 0.1 points on the 1.0-10.0 scale, the spectral learner can do better than random guessing, and its accuracy improves steadily as the distance between scores increases.

6.2 Background

Before describing the spectral learning approach, some background in Fourier analysis and Fourier-based learning are needed.

6.2.1 Spectral (Fourier) Analysis

Suppose f is a real function of n Boolean inputs (i.e., $f : \{0,1\}^n \rightarrow \mathbb{R}$). Then the Fourier representation of f, denoted $\hat{f}$, is given by

$$\hat{f}(\alpha) = \frac{1}{2^n} \sum_{x \in \{0,1\}^n} f(x)\chi_\alpha(x) \tag{6.1}$$

where $\alpha \in \{0,1\}^n$ and χ_α is a Fourier basis function defined by the following:

$$\chi_\alpha(x) = (-1)^{\sum_i \alpha_i x_i} = \begin{cases} +1 : \text{if } \sum_i \alpha_i x_i \text{ is even} \\ -1 : \text{if } \sum_i \alpha_i x_i \text{ is odd} \end{cases} \tag{6.2}$$

Each $\hat{f}(\alpha)$ is a Fourier coefficient whose magnitude is proportional to the correlation between f and χ_α. Any f can be expressed in terms of the Fourier basis functions by the following, which shows how a function can recovered from its Fourier representation:

$$f(x) = \sum_{\alpha \in \{0,1\}^n} \hat{f}(\alpha)\chi_\alpha(x) \tag{6.3}$$

The Fourier basis functions are XOR functions, each returning -1 if and only if the XOR of a particular subset of the inputs is true. The subset is implicitly defined by the binary digits of α, and it is the set $S = \{i : \alpha_i = 1\}$. (Note that $(\alpha_i = 0) \Rightarrow (\alpha_i x_i = 0)$ and $(\alpha_i = 1) \Rightarrow (\alpha_i x_i = x_i)$, so the output of each χ_α can be determined from those inputs from which $\alpha_i = 1$.) The basis functions can be defined in terms of S, leading to the following:

$$\chi_S(x) = (-1)^{\sum_{i \in S} x_i} = \begin{cases} +1 : \text{if } \sum_{i \in S} x_i \text{ is even} \\ -1 : \text{if } \sum_{i \in S} x_i \text{ is odd} \end{cases} \tag{6.4}$$

The order of a basis function χ_α or χ_S is given by $|\{i : \alpha_i = 1\}|$ or $|S|$, respectively, and it is the number of inputs over which the basis function applies its Boolean operator.

The Fourier basis provides just one possible representation for a function. For example, f could be represented in terms of basis functions that compute ANDs or ORs of the inputs. Patterned after the Fourier basis functions, AND (ξ) and OR (ζ) basis functions can be defined as follows:

$$\xi_S(x) = \begin{cases} +1 : \text{if } \sum_{i \in S} x_i < |S| \\ -1 : \text{if } \sum_{i \in S} x_i = |S| \end{cases} \tag{6.5}$$

$$\zeta_S(x) = \begin{cases} +1 : \text{if } \sum_{i \in S} x_i = 0 \\ -1 : \text{if } \sum_{i \in S} x_i > 0 \end{cases} \tag{6.6}$$

These basis functions can be substituted into Equation 6.2 to obtain coefficients that indicate the correlation between f and the ANDs or ORs of the inputs. Note however, that these coefficients do not generally give the linear combination of AND or OR functions that is equivalent to f. (There is a different transformation to obtain those coefficients, but it will not be of interest here.)

6.2.2 Spectral Learning

Spectral learning algorithms attempt to learn f by learning a spectral representation, $\hat{f}$, of f. The most common approach is to attempt to approximate $\hat{f}$ from a set X of $\langle x, f(x) \rangle$ examples. Typically, the spectral coefficients are approximated as follows:

$$\tilde{\hat{f}}(\alpha) = \frac{1}{|X|} \sum_{\langle x, f(x) \rangle \in X} f(x) \phi_\alpha(x) \tag{6.7}$$

(Here and in the following we will use ϕ_α and ϕ_S as generic basis function references that could refer to any of the previously introduced basis functions.) For a function of n inputs, there are 2^n spectral coefficients and basis functions. Consequently, it is not possible to use all basis functions unless n is small. Therefore, spectral learning algorithms that attempt to estimate $\hat{f}$ will generally use only the basis functions whose coefficients are largest (in magnitude). If A is the set of labels of those basis functions, then a spectral learner's

representation of f is given by

$$\tilde{f}(x) = \sum_{\alpha \in A} \tilde{\hat{f}}(\alpha)\phi_\alpha(x) \tag{6.8}$$

Since there are an exponential number of coefficients, it is impractical to do a brute-force search for large coefficients. Furthermore, certain classes of functions can be approximated well by representations that include only low-order basis functions. Consequently, some spectral learning algorithms limit their attention to low-order basis functions and implicitly assume that the coefficients of higher-order basis functions can be ignored [Klivans et al., 2004, Kargupta and Park, 2004, Linial et al., 1993]. Empirical results have shown that low-order spectral models often outperform models that attempt to include higher-order basis functions with large coefficients [Drake and Ventura, 2011a].

One alternative approach to spectral learning is based on boosting [Jackson, 1997, Jackson et al., 2002]. In the classification setting in which this approach was introduced, the basis functions are thought of as weak hypotheses (i.e., approximations of f that may do only slightly better than random guessing) that are combined to form a strong hypothesis. Basis functions are selected iteratively based on correlation with a weighted version of the training data. Initially, all examples are given equal weight, and the first basis function is selected based on correlation with the original data. After each basis function is added, however, examples that were misclassified are given more weight, such that over time the algorithm focuses more on examples that the current model (i.e., the current set of basis functions and coefficients) misclassifies.

6.3 A New Spectral Learning Algorithm

The core spectral learning algorithm used in this paper is a new spectral algorithm, specifically designed for natural language problems, that combines the low-order and boosting approaches to spectral learning. The algorithm is essentially a boosting algorithm, but the

LearnSpectralModel(X, k)

(1) $M \leftarrow \left\{\left\langle \phi_\emptyset, \frac{1}{|X|}\sum_X f(x)\phi_\emptyset(x)\right\rangle\right\}$

(2) $C \leftarrow \{\phi_S : |S| = 1\}$

(3) for $i = 1$ to k

(4) $X_i \leftarrow \{\langle x, e(x)\rangle : \langle x, f(x)\rangle \in X \wedge e(x) = f(x) - \sum_{\langle \phi_S, \tilde{\hat{f}}(S)\rangle \in M} \tilde{\hat{f}}(S)\phi_S(x)\}$

(5) $\phi_T \leftarrow \text{argmax}_{\phi_S \in C} \frac{1}{|X_i|}\sum_{X_i} e(x)\phi_S(x)$

(6) $M \leftarrow M \cup \left\{\left\langle \phi_T, \frac{1}{|X_i|}\sum_{X_i} e(x)\phi_T(x)\right\rangle\right\}$

(7) $C \leftarrow C \setminus \{\phi_T\}$

(8) if $|T| = 1$

(9) for each $\langle \phi_S, \tilde{\hat{f}}(S)\rangle \in M$ s.t. $|S| = 1$

(10) $C \leftarrow C \cup \{\phi_{S \cup T}\}$

(11) return M

Figure 6.1: The core spectral learning/feature selection algorithm. Given a set X of examples of a target function f, the algorithm returns a spectral model (consisting of $k + 1$ basis functions and their corresponding coefficients) that approximates f. The algorithm can also be used as a feature selector, as the selected basis functions can be used as input features to another learning algorithm.

set of basis functions that can be added to the model each iteration is restricted in a way that forces the learner to use low-order basis functions before high-order functions. This core algorithm is used by a multi-spectrum meta-algorithm that builds a spectral model in each of the AND, OR, and XOR representations and then combines them into a single model.

The core algorithm, shown in Figure 6.1, always begins by adding the 0^{th}-order basis function and the corresponding coefficient (computed via Equation 6.2) to the model M (line 1). (The empty set symbol, $\emptyset$, is used as the label for this basis function since its set S is empty.) After adding the 0^{th}-order basis function to the model, the algorithm initializes a set C of candidate basis functions with all of the 1^{st}-order basis functions (line 2).

Then, the algorithm enters a loop (line 3) that continues until k additional basis functions have been added to the model. In each iteration of the loop, the set of training examples, X, is converted into a modified training set, X_i, in which the examples' outputs are set to the residual error of the model (i.e., to the difference between f and the current output of the model) (line 4). Then, the basis function ϕ_T in C that is most correlated

with this residual error is selected (line 5) and added to the model (line 6). (Note that the coefficient for ϕ_T is computed with respect to X_i, not X, so that it is weighted optimally with respect to reducing the remaining error.) Basis function ϕ_T is then removed from C (line 7).

The final step in each iteration of the main loop is to add additional candidate basis functions to C, if necessary (lines 8-10). The algorithm does not allow a basis function ϕ_S to be added to C until the basis functions representing all subsets of S have been added. Thus, for example, the 2^{nd}-order function $\xi_{\{3,10\}}$, that computes the AND of inputs 3 and 10, will not be added to C until the 1^{st}-order basis functions $\xi_{\{3\}}$ and $\xi_{\{10\}}$ have been added. The algorithm presented in Figure 6.1 is simplified slightly from this general case, as it assumes that only 2^{nd}- or lower-order basis functions should be considered for the model. (This restriction was used to obtain the results that follow as well.) Therefore, if ϕ_T is a 1^{st}-order basis function (line 8), then M is searched for previously added 1^{st}-order basis functions (line 9), and the 2^{nd}-order basis functions that represent the union of the 1^{st}-order basis functions' subsets are added to C (line 11). If ϕ_T is a 2^{nd}-order basis function, then no additional basis functions are added to C.

This algorithm combines the benefits of a low-order approach (such as good generalization) with the additional representational power of higher-order features. Since a basis function is considered for inclusion only after the lower-order basis functions for its subset of inputs have been added, and since the iterative boosting approach selects basis functions based on correlation with the residual error in the model, the algorithm tends to add a higher-order basis function only when there is useful information in the higher-order feature that was not captured by the lower-order features.

As mentioned previously, the core spectral learning algorithm described in Figure 6.1 is the main component of a meta-algorithm for spectral learning. Specifically, the meta-algorithm executes the core spectral learning algorithm three times, once for each of the AND, OR, and XOR bases, and the final model averages the individual spectral models. If

A_ξ, A_ζ, and A_χ represent the sets of labels of the basis functions selected while learning the AND, OR, and XOR representations, respectively, then the final model can be expressed by the following:

$$\tilde{f}(x) = \frac{1}{3}\left(\sum_{\alpha \in A_\xi} \tilde{\hat{f}}(\alpha)\xi_\alpha(x) + \sum_{\alpha \in A_\zeta} \tilde{\hat{f}}(\alpha)\zeta_\alpha(x) + \sum_{\alpha \in A_\chi} \tilde{\hat{f}}(\alpha)\chi_\alpha(x)\right) \tag{6.9}$$

By combining models built in each representation, the spectral learner is able to take advantage of useful AND, OR, and XOR features. (Note that it is possible to directly combine the three types of higher-order features by considering all three types while building a single model. However, previous work with multi-spectrum models suggests that building the models separately improves generalization and decreases the likelihood of overfitting [Drake and Ventura, 2011b].)

6.4 Spectral Learning Results

As stated in the introduction, our goal is to build a system for assigning sentiment scores to products based on a written evaluation (e.g., a product review). Our current application domain is video games, and the task is to assign a score on a 1.0-10.0 scale that captures the overall sentiment of the author of a review with respect to a particular game. The data that we use in the following is a collection of 4,972 reviews from GameSpot.com, each of which has an associated rating on a 1.0-10.0 scale with 0.1-point increments. This set of reviews represents all reviews posted on the site from January 2002 through December 2006. These reviews were contributed by 70 different authors, although some authors contributed many more reviews than others. The average length of the reviews (in terms of the number of words) is 1,347, although the length varies considerably. For example, the smallest review contains only 98 words, while the largest contains 5,194 words.

We use a word-presence document model in which each review is reduced to a binary vector x of length n such that $x_i = 1$ if word i occurs in the review and $x_i = 0$ otherwise.

The n words that correspond to the n indices of x are selected from the training data. Since the spectral learning algorithm performs implicit feature selection, all words observed in the training data could be used in the word presence vectors, even though most words would occur so infrequently (or so frequently) that the corresponding 1^{st}-order basis functions would be unlikely to ever be added to the model. However, to improve computational efficiency, the set of all observed words (about 45,000) was reduced to the 2,000 words that had the highest 1^{st}-order correlation with the training data after subtracting the mean value (i.e., average rating) from the function. (Subtracting the average value of the function ensures that coefficients that are large in magnitude indicate words that are predictive of above-average or below-average ratings; otherwise, coefficient size is not necessarily indicative of predictive power.) This word-selection method can be expressed formally by the following:

$$\underset{\{w_1,\ldots w_n\}\in W}{\operatorname{argmax}} \sum_{w\in\{w_1,\ldots w_n\}} \left| \frac{1}{|R|} \sum_{r\in R} \bar{f}(r) I_w(r) \right| \tag{6.10}$$

in which R is the set of training reviews, W is the set of words in those reviews, $I_w(r)$ is an indicator function that outputs 1 if review r contains word w and outputs -1 otherwise, and $\bar{f}(r)$ is the rating associated with training review r (after subtracting the average rating from all reviews).

To test the performance of the learning methods, we used a leave-one-year-out cross-validation approach. That is, one year of data was held out, the learner was training on the remaining data (approx. 4,000 reviews), and then the learner was tested on the held out portion. This process was repeated five times, once for each possible test year, and results were averaged over those five trials.

Table 6.1 shows the average absolute difference between the spectral learner's sentiment scores and the actual ratings associated with the test reviews. For comparison, results obtained by applying linear regression and linear support vector machine [Joachims, 1999] models are also shown, as is a baseline performance measure that indicates the performance

Table 6.1: Average prediction error of the spectral, SVM, and linear regression learners, as well as the prediction error of a baseline learner. The spectral learner's sentiment scores differed from the actual ratings by less than 0.75 points on average (a 36.6% reduction in baseline error).

Algorithm	Ave Prediction Error
Spectral Learner	0.746
SVM	0.764
Linear Regr	0.819
Baseline	1.176

obtained when using the mean score observed in the training data as the prediction for each test review. (The baseline error is included primarily to emphasize the non-uniformity in the distribution of ratings. If the ratings were uniformly distributed, this baseline error would be approximately 2.25; however, the distribution is not uniform and it is skewed positively around an average of about 6.9.)

The sentiment scores assigned by the spectral learner differed by only 0.746 points on average from the actual scores associated with the reviews. This corresponds to a 36.6% reduction in error relative to the baseline. The spectral learner also performed significantly better (statistically) than the linear regression model. (Statistical significance was measured by a paired permutation test, and significant differences were identified as those for which $p \leq 0.05$.) In addition, the spectral learner outperformed the SVM learner on average, although this difference was not statistically significant.

Table 6.2 demonstrates the benefits of the "ensemble" spectral learning approach, as it compares the performance of a spectral learner that combines the AND, OR, and XOR bases with spectral learners that use just one those bases. The spectral learner that combines the three bases and can take advantage of each higher-order feature type performs significantly better (statistically) than any of the single-basis spectral learners.

Table 6.2: Average prediction error of a spectral learner that uses an ensemble of AND, OR, and XOR representations, compared to the error of spectral learners that use just one of those representations. The multi-spectrum approach significantly improves (statistically) the accuracy of the spectral learner.

Algorithm	Ave Prediction Error
Spectral Learner	0.746
Spectral Learner (AND only)	0.776
Spectral Learner (OR only)	0.783
Spectral Learner (XOR only)	0.811

6.5 Spectral Features

The 0^{th}- and 1^{st}-order basis functions of the AND, OR, and XOR bases are all identical. The 0^{th}-order basis function is a constant function, so it functions like an intercept in linear regression or like a "bias" input in neural networks. In the spectral learning method presented previously, it is always the first basis function added to the model. The coefficient assigned to it is always the mean value of the function being learned, so it causes the next spectral feature to be selected based on how well it models deviations from the mean.

There are n 1^{st}-order basis functions in each basis, and each indicates whether a specific input's value is true or false. Specifically, basis function $\phi_{\{i\}}$ outputs -1 if x_i is true and outputs $+1$ if it is false; thus, the 1^{st}-order basis functions simply map the original Boolean input features from $\{0, 1\}$ to $\{1, -1\}$.

Since the spectral coefficients (computed by Equation 6.2) are proportional to the correlation between f and each basis function, the 1^{st}-order spectral coefficients provide a natural way of determining which words are predictive of above average or below average sentiment. In addition, unlike other word selection methods that are based on frequency (e.g., tf-idf) of classes (e.g., mutual information), the spectral coefficients can be applied naturally to and take advantage of documents labeled by numeric scores.

The sentiment word selection method implied by Equation 6.10 is equivalent to selecting the n words that have the largest corresponding 1^{st}-order spectral coefficients (after

Table 6.3: The ten largest positive and negative 1st-order spectral coefficients and the corresponding words. The 1st-order spectral coefficients provide a natural and effective method for identifying sentiment words.

Positive		Negative	
0.537	great	-0.201	bad
0.365	excellent	-0.196	decent
0.359	new	-0.185	dull
0.338	quite	-0.182	poor
0.322	addition	-0.178	poorly
0.305	best	-0.176	bland
0.300	keep	-0.157	worse
0.296	nice	-0.151	repetitive
0.292	previous	-0.150	terrible
0.292	features	-0.150	generic

subtracting the average score from the function). For the data used in this paper, the ten words with the largest positive and negative spectral coefficients are shown in Table 6.3.

The AND, OR, and XOR bases differ in their 2nd- and higher-order basis functions, and it is these spectral features that represent an extension of the word presence model. As the following section will show, these features allow other learning algorithms to achieve better performance. Since the spectral algorithm presented in this paper does not allow a 2nd-order basis function $\phi_{\{i,j\}}$ to be added to the model unless the 1st-order basis functions $\phi_{\{i\}}$ and $\phi_{\{j\}}$ have been added, and since the basis functions are selected iteratively based on correlation with the residual error of previously added basis functions, the 2nd-order basis functions that are added to the model can be thought of as providing additional information about f that could not be captured by the 1st-order features alone.

For example, the 2nd-order AND basis functions allow the model to account for the fact that when both of two sentiment-carrying words are present, the combined effect may be different than the sum of the individual contributions of those words. For example, in all but one of the trials of the cross-validation experiments that involved the AND basis, the word "great" was the first feature selected by the algorithm, the word "new" was the

third selected, and the 2nd-order AND feature ("great" ∧ "new") was about the thirteenth feature selected. The words "great" and "new" were both positively correlated with ratings, and their coefficients of about 0.5 and 0.3 indicated that the model's predicted score should increase by 0.5 and 0.3, respectively, when those words are present. However, the ("great" ∧ "new") feature was negatively correlated with the function when it was added, and the coefficient was about -0.1, indicating that when both words are present the prediction should increase by only 0.7 $(0.5 + 0.3 - 0.1)$, and not by 0.8.

The 2nd-order OR and XOR basis functions allow for slightly different types of additional information to be factored into the model. The OR features allow a learner to model the effect of observing "either one or both" of two sentiment-carrying words, while the XOR features allow a learning algorithm to model the effect of observing "either one but not both" of two sentiment-carrying words. The AND, OR, and XOR features are similar, but they each provide unique information, and they extend the word presence model in ways that allow for more accurate sentiment analysis.

6.6 Using Spectral Features to Improve Learning

Although the spectral algorithm presented in Figure 6.1 produces a model that can be used directly for assigning sentiment scores to documents, the algorithm can also be used as a feature selection method, as the selected spectral features can be used as input features to other learning algorithms.

Table 6.4 shows how the performance of the linear regression and SVM learners improves when they use spectral features selected by the algorithm in Figure 6.1. Specifically, the "w/ Spectral Features" results were obtained by building three linear regression or SVM models (one with the spectral features obtained from the AND basis, one with the features obtained from the OR basis, and one with the features obtained from the XOR basis), and then averaging the predictions of the models as was done previously for the spectral learner. The average prediction error of both algorithms decreases when using this spectral approach,

Table 6.4: Average prediction error of linear regression and SVM models that are trained with the original input features and of linear regression and SVM models that are "averaged" over models trained with the AND, OR, and XOR spectral features. Both algorithms perform significantly better (statistically) when using the spectral feature approach.

Algorithm	Average Prediction Error
SVM	0.764
SVM w/ Spectral Features	0.731
Linear Regr	0.819
Linear Regr w/ Spectral Features	0.749

and the differences are statistically significant in both cases. When using the spectral approach, the SVM algorithm outperforms the spectral learner's average performance (0.746, in Table 6.1), although that difference is not statistically significant.

6.7 Identifying Relative Differences in Sentiment

Figure 6.2 presents results on a related sentiment task: identifying which of two products is better (overall) based on written reviews for the products. The graph shows how often the SVM learner (when using the spectral feature approach) was able to correctly determine from two reviews which game had the higher rating, as a function of the difference in rating between the two games. Even at the smallest possible difference in ratings (0.1 points), the learner is able to do better than randomly guessing. It correctly identified the game with the higher rating 53.1% of the time, which is a statistically significant improvement over a learner that randomly guesses. And, as the difference in rating increases, the performance of the learner steadily improves. When the ratings differ by just 1.0 points on the 1.0-10.0 scale, the learner's accuracy is 75.3%; when the difference is 2.0 points, the accuracy increases to 90.1%.

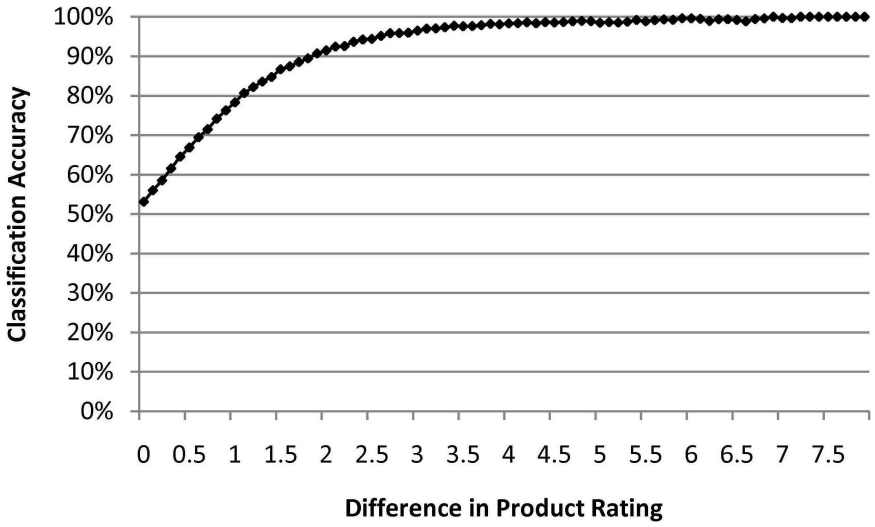


Figure 6.2: The accuracy of an SVM learner (when using the spectral feature approach) at identifying which of two products had a higher rating, as a function of the difference between the ratings. Even when the products' ratings differ by only 0.1 points on the 1.0-10.0 scale, the learner does significantly better (statistically) than randomly guessing, and its performance improves steadily as the difference between ratings increases.

6.8 Conclusion

Determining the sentiment that is expressed in written text is an important sentiment analysis task. In this paper, we have considered the problem of assigning an overall sentiment score to a document. A common approach to this type of problem is to identify a set of sentiment words and then to use machine learning to learn to classify document sentiment based on the presence/absence of those words. In this paper, we have presented an extension of this word presence approach that is based on spectral representations. We have introduced a new spectral learning algorithm that implicitly identifies words, as well as higher-order functions of those words, that are useful for predicting overall sentiment. The higher-order functions extend the word presence model by giving the learner access to information about how ANDs, ORs, and XORs of the word presence features affect overall document sentiment. The spectral features can be used directly in a spectral sentiment model, but they can also be used as input features to any learning algorithm. In contrast to previous unsuccessful at-

tempts to improve the word presence model, these features are shown to provide statistically significant improvements in performance.

There are several directions for future work that could potentially lead to improved performance. For example, no attempt has been made here to distinguish between objective and subjective statements, or between opinions that refer directly to the product being reviewed and those that refer to something else (e.g., another product), and it may be possible to improve performance by recognizing these differences. Another interesting area for future work will be to determine if this spectral learning approach is equally beneficial in related domains, such as topic classification. Finally, the results in Figure 6.2 show that it is possible to identify very small differences in sentiment with better-than-random accuracy. An interesting direction for further study will be to address the question of how finely sentiment can be modeled.

Chapter 7

The Maximum Satisfiability and Largest Coefficient Problems

Abstract

Recently, an interesting link between spectral learning and the maximum satisfiability problem (MAX-SAT) was established. Specifically, it was shown that MAX-2-SAT can be reduced to the problem of finding the largest coefficient in a spectral representation, which is a central problem in spectral learning. In this paper, we explore this connection, as we apply coefficient finding methods to MAX-SAT problems, and apply MAX-SAT techniques to the problem of finding large spectral coefficients.

7.1 Introduction

Fourier analysis has become a powerful tool in theoretical machine learning, particularly for proving learnability results. These results usually focus on the spectral characteristics of a class of functions, and positive results require a demonstration that these characteristics admit efficient computation. Since the full Fourier spectrum is exponential in the size of the input, one requirement for efficiency is that a small subset of the spectral coefficients be "large" (in other words, the function class must be closely approximated with a small number of basis functions). Also, since computing the full spectrum is $O(n2^n)$, some efficient algorithm for determining *which* of the coefficients are large is required. This can be difficult, and historically two approaches have been taken: 1) choose a class of functions that admits a simple (often brute force) search of a small subset of the coefficients, or 2) admit the use of an oracle.

Recently, it was shown that the problem of finding a large Fourier coefficient is NP-complete by reduction from MAX-2-SAT (see Chapter 2). This hardness result is not specific to the Fourier basis, as similar results were shown for bases of AND and OR functions.

The reduction from MAX-2-SAT reveals the similarity between satisfiability problems and the problem of finding large coefficients in a spectral representation. Based on this observation, we apply a search algorithm developed for finding large spectral coefficients to MAX-SAT problems, and we apply techniques used in state-of-the-art MAX-SAT solvers to the problem of finding the largest Fourier coefficient.

7.2 Background and Definitions

Let f be any function of the form $f : \{0,1\}^n \longrightarrow \mathbb{R}$. Then the Fourier spectrum of f, denoted $\hat{f}$, is given by

$$\hat{f}(\alpha) = \frac{1}{2^n} \sum_{x \in \{0,1\}^n} f(x)\chi_\alpha(x) \tag{7.1}$$

where $\alpha \in \{0,1\}^n$ and $\hat{f}(\alpha)$ is the spectral coefficient corresponding to basis function $\chi_\alpha : \{0,1\}^n \rightarrow \{-1,1\}$, which is defined as follows:

$$\chi_\alpha(x) = \begin{cases} 1 : \text{if } \sum_i \alpha_i x_i \text{ is even} \\ -1 : \text{if } \sum_i \alpha_i x_i \text{ is odd} \end{cases}$$

When weighted by their coefficients, a linear combination of the basis functions gives the original function:

$$f(x) = \sum_{\alpha \in \{0,1\}^n} \hat{f}(\alpha)\chi_\alpha(x) \tag{7.2}$$

The Fourier transform described above applies to functions of Boolean inputs and represents a special case of the discrete Fourier transform that is commonly referred to as a Walsh transform.

In typical learning scenarios, f is unknown, and $\hat{f}$ must be approximated from a set X of $\langle x, f(x)\rangle$ pairs. The approximate Fourier spectrum of f, obtained from X, which we will denote $\hat{f}_X$, is given by the following:

$$\hat{f}_X(\alpha) = \frac{1}{|X|} \sum_{\langle x, f(x)\rangle \in X} f(x)\chi_\alpha(x) \tag{7.3}$$

We will also refer to $\hat{f}_X$ as the Fourier spectrum of X. To be useful in a learning (function approximation) context, the approximate spectrum must not differ too much from the exact spectrum; that is, we require $\hat{f}_X(\alpha) \approx \hat{f}(\alpha)$ for $|X|$ of reasonable size. Fortunately, a well-known result [Mansour, 1994] guarantees that this is the case.

The basis functions of the Fourier transform are XOR functions, each computing the XOR of a different subset of the inputs. (The subset is determined by α, as any input for which $\alpha_i = 0$ is ignored.) Logical AND and OR relationships have been shown to be useful in many learning settings, and spectral representations based on AND and OR functions can be derived using similarly defined basis functions, $\xi_\alpha : \{0,1\}^n \rightarrow \{-1,1\}$ and

$\zeta_\alpha : \{0,1\}^n \rightarrow \{-1,1\}$:

$$\xi_\alpha(x) = \begin{cases} 1 : \text{if } \sum_i \alpha_i x_i < \sum_i \alpha_i \\ -1 : \text{if } \sum_i \alpha_i x_i = \sum_i \alpha_i \end{cases}$$

$$\zeta_\alpha(x) = \begin{cases} 1 : \text{if } \sum_i \alpha_i x_i = 0 \\ -1 : \text{if } \sum_i \alpha_i x_i > 0 \end{cases}$$

Equation 7.3 can be used to compute approximate spectral coefficients for these representations by replacing the Fourier basis functions with either AND or OR functions. We will refer to these spectra as the AND and OR spectra of X. Note that Equation 7.2 does not provide an inverse transform for these representations (i.e., these coefficients do not generally give a linear combination for representing f); however, as in the Fourier case, these coefficients measure the correlation between each basis function and f.

We now give definitions for several decision problems (cast as set membership problems). The first is a well-known NP-complete problem [Garey and Johnson, 1979].

Definition 8 (MAX-SAT)**.** *Given a set U of binary variables, a set C of CNF clauses over U, and a scalar m, $0 < m \leq |C|$,* MAX-SAT $= \{\langle U, C, m\rangle |$ *there exists a truth assignment for the variables in U that simultaneously satisfies at least m clauses in $C\}$.*

The next is a problem of interest in the theoretical machine learning community [Linial et al., 1989, Kushilevitz and Mansour, 1993, Jackson, 1997] that has more recently become important in applied settings as well [Mansour and Sahar, 2000, Drake and Ventura, 2005].

Definition 9 (LARGE-FOURIER-COEF)**.** *Given a scalar $n \in \mathbb{Z}$, a set X of $\langle x, f(x)\rangle$ pairs where $x \in \{0,1\}^n$ and $f(x) \in \mathbb{R}$, and a scalar $p \in \mathbb{R}$,* LARGE-FOURIER-COEF $= \{\langle n, X, p\rangle |$ *there exists $\alpha \in \{0,1\}^n$ such that $\left|\hat{f}_X(\alpha)\right| \geq p$, where $\hat{f}_X$ is the Fourier spectrum of X (Equation 7.3)*$\}$.

Simple variations on the LARGE-FOURIER-COEF problem can be constructed by considering other bases. Here, we define variations for the OR and AND bases.

Definition 10 (LARGE-OR-COEF)**.** *Given a scalar $n \in \mathbb{Z}$, a set X of $\langle x, f(x)\rangle$ pairs where $x \in \{0,1\}^n$ and $f(x) \in \mathbb{R}$, and a scalar $p \in \mathbb{R}$,* LARGE-OR-COEF $= \{\langle n, X, p\rangle |$ *there exists $\alpha \in \{0,1\}^n$ such that $\left|\hat{f}_X(\alpha)\right| \geq p$, where $\hat{f}_X$ is the OR spectrum of X}$\}$.

Definition 11 (LARGE-AND-COEF)**.** *Given a scalar $n \in \mathbb{Z}$, a set X of $\langle x, f(x)\rangle$ pairs where $x \in \{0,1\}^n$ and $f(x) \in \mathbb{R}$, and a scalar $p \in \mathbb{R}$,* LARGE-AND-COEF $= \{\langle n, X, p\rangle |$ *there exists $\alpha \in \{0,1\}^n$ such that $\left|\hat{f}_X(\alpha)\right| \geq p$, where $\hat{f}_X$ is the AND spectrum of X}$\}$.

7.3 Using Coefficient Search Techniques to Solve Satisfiability Problems

The NP-Completeness result in Chapter 2 regarding the hardness of finding large spectral coefficients shows that an instance of the MAX-2-SAT problem can be converted easily into an instance of the problem of finding the largest spectral coefficient. Based on this connection between the satisfiability and large coefficient problems, we now show how a simple modification to an existing algorithm for finding large spectral coefficients allows it to solve SAT and MAX-SAT problems.

7.3.1 Algorithm

We derive a SAT solver from the spectral coefficient algorithm presented in [Drake and Ventura, 2005]. The algorithm, which we will refer to as the BFS algorithm, uses a best-first search with backtracking to explore the set of possible basis function labels and find the $\alpha \in \{0,1\}^n$ corresponding to the largest spectral coefficient.[1]

The algorithm begins with an undefined label $\alpha = *^n$ ($*$ denotes undefined) and then expands its search frontier by assigning values to the digits of α. The order in which digits are assigned is determined dynamically so as to reduce the total required search size. For

[1]The algorithm can continue beyond the largest coefficient to find the k largest coefficients; however, finding the single largest coefficient will be sufficient for our SAT solver.

each partially-defined label, a bound on $\hat{f}(\alpha)$ is computed that applies to all α that could result from future digit assignments. Using this bound, exploration continues in best-first order until a fully defined label is found. At that point the algorithm has found the largest coefficient.

An algorithm for solving SAT problems can be easily created from this spectral coefficient algorithm. The overall search structure remains the same: search over $\{0,1\}^n$ in a best-first manner until an optimal solution is found. For SAT or MAX-SAT, the n-bit binary labels can represent truth assignments to the n variables. All that must change is the method for computing upper bounds on the utility of partially defined labels.

We accomplish this as follows: for any label $\alpha \in \{0,1,*\}^n$, we define the utility of α, $U(\alpha)$, to be $|C| - m$, where C is the set of clauses and m is the number of clauses that cannot be satisfied by any truth assignment that could be derived from α by future digit assignments. Given α, a clause c can be labeled as unsatisfiable if it is not satisfied by the partial assignment and no literal in c corresponds to a variable whose assignment has not been defined. Notice that if this algorithm is used as a SAT solver instead of a MAX-SAT solver, any α for which $U(\alpha) \leq |C|$ can immediately be pruned from the search.

Like the spectral coefficient algorithm from which this MAX-SAT solver is derived, it is an exact algorithm, always returning the correct solution; however, it comes with no guarantee of termination within a feasible amount of time (or memory). A crucial variable in its performance is the number of partially and fully defined labels considered before arriving at a solution. At best, the algorithm will consider n labels. This will occur only when the initial assignments made to the variables lead to an optimal solution and no backtracking is required. The worst-case result can be exponential, however, as there are 2^n potential labels to consider.

7.3.2 Empirical Results

We now present results of applying this algorithm to several SATLIB benchmarks [Hoos and Stützle, 2000, Cheeseman et al., 1991]. Table 7.1 summarizes algorithm performance on these benchmarks, and for each set of benchmarks shows the smallest, average, and largest search sizes (number of considered labels) required to either determine satisfiability or find the maximum number of satisfiable clauses. Test cases labeled (sat) and (unsat) represent satisfiable and unsatisfiable test cases, respectively, and the corresponding results show the search sizes required to determine satisfiability. For unsatisfiable test cases, the search size required to find the exact MAX-SAT solution is also shown. Those test cases are labeled (maxsat).

As expected, the size of the search tends to grow as the number of variables increases. Also as expected, the required search size tends to be larger for unsatisfiable instances than for satisfiable instances, while for unsatisfiable instances, the search size is larger when finding the exact MAX-SAT solution than when only determining whether there exists a truth assignment that satisfies all clauses. Even in the largest searches, however, the number of considered labels remains very small relative to the size of the search space. For example, for the worst-case result of any 50-input problem, the fraction of possible (partial) truth assignments considered is $169,488/2^{50} \approx 1.5 \times 10^{-10}$, while the same fraction for the worst-case 75-input problem is $2,818,265/2^{75} \approx 7.5 \times 10^{-17}$.

Results marked with * in Table 7.1 indicate cases for which some instances could not be included in the results due to memory constraints on our test machine. That is, the search frontier became too large to hold entirely in memory while finding an exact solution. Thus, had all cases been considered, the average and worst search sizes would be larger. This represents the primary weakness of the algorithm, as its ability to find an exact solution quickly requires that a large search frontier be stored in memory, limiting the size of problems it can solve. The memory efficiency of the algorithm could likely be improved

Table 7.1: Results of using a SAT/MAX-SAT solver derived from an algorithm for finding large spectral coefficients. For each set of benchmarks, the smallest (best), average, and largest (worst) search sizes required to either determine satisfiability or find the maximum number of satisfiable clauses is shown. Satisfiable test cases (sat) are separated from unsatisfiable cases (unsat). For unsatisifiable cases, the search sizes required to find exact MAX-SAT solutions are also reported (maxsat).

Benchmark	Variables	Clauses	Instances	Best	Average	Worst
uf20-91 (sat)	20	91	1,000	20	87	505
uf50-218 (sat)	50	218	1,000	50	2,654	35,248
uuf50-218 (unsat)	50	218	1,000	1,051	7,139	55,001
uuf50-218 (maxsat)	50	218	1,000	1,132	13,107	169,488
uf75-325 (sat)	75	325	100	121	36,619	188,688
uuf75-325 (unsat)	75	325	100	14,566	91,921	317,912
uuf75-325 (maxsat)	75	325	100	14,912	237,982	2,818,265
uf100-430 (sat)	100	430	992	111	414,343*	3,816,257*
uuf100-430 (unsat)	100	430	945	66,746	1,168,251*	3,852,729*
uuf100-430 (maxsat)	100	430	389	140,176	1,226,704*	3,852,736*
jnh1-20 (sat)	100	850	4	218	3,125	7,493
jnh1-20 (unsat)	100	850	16	4,989	95,258	1,019,358
jnh1-20 (maxsat)	100	850	16	20,622	749,368	3,249,544
jnh201-220 (sat)	100	800	11	100	123,794	913,153
jnh201-220 (unsat)	100	800	9	3,048	69,133	264,311
jnh201-220 (maxsat)	100	800	9	41,883	120,111	264,321
jnh301-310 (sat)	100	900	1	768,653	768,653	768,653
jnh301-310 (unsat)	100	900	9	112	33,520	216,588
jnh301-310 (maxsat)	100	900	5	199,738	1,403,510*	3,119,589*

through additional implementation tricks, while pruning methods could be introduced to drastically improve memory efficiency in exchange for correctness guarantees.

7.4 Using MAX-SAT Techniques to Find Large Spectral Coefficients

In the previous section, a coefficient search algorithm was used to solve SAT and MAX-SAT problems. However, the similarity between the large coefficient and satisfiability problems also suggests that algorithms and theoretical results developed for SAT problems may have relevant application in spectral learning. We now provide an example of such an application, as we apply MAX-SAT techniques to the problem of finding large Fourier coefficients.

7.4.1 MAX-SAT Techniques and Finding Coefficients

Many popular algorithms for solving satisfiability problems are based on the Davis-Putnam-Loveland-Logemann procedure [Davis and Putnam, 1960, Davis et al., 1962]. In its basic form, this method performs a recursive branch-and-bound search through the space of possible truth assignments, pruning branches that cannot possibly lead to better solutions than the best found so far. A coefficient finding algorithm could explore the space of basis function labels in a similar fashion. This approach has been used before [Kushilevitz and Mansour, 1993, Mansour and Sahar, 2000], although an oracle was used by those algorithms to guide the search efficiently.

The BFS algorithm used in the previous section performs a best-first search through the space of basis function labels and does not require an oracle. The algorithm can potentially find the largest Fourier coefficients very quickly. However, because it must store the entire search frontier in memory, its space complexity is exponential in the worst case ($O(2^n)$). Consequently, memory limitations will prevent its use in some domains.

Here, we compare the performance of the BFS algorithm to a similar algorithm that is based on the recursive branch-and-bound search often used in SAT algorithms. We shall refer to the branch-and-bound approach as the B&B algorithm.[2]

In addition to applying a recursive branch-and-bound search, we investigate two techniques that have improved the performance of SAT algorithms [Borchers and Furman, 1998, Alsinet et al., 2003]. One of these techniques is the separation of the problem into two phases. In the first phase, a greedy algorithm, such as WalkSAT [Jiang et al., 1995], is used to find a "good" solution to the problem. Then, in the second phase, the branch-and-bound procedure described previously is run, using the "good" solution as an initial bound for pruning. We use a simple variation on WalkSAT, which we call WalkCoef, that walks through the same $\{0,1\}^n$ space, but in search of large coefficients rather than truth assignments.

The second technique is the use of a heuristic for determining the order in which variables should be considered during search. One heuristic that has been effective for MAX-SAT problems is the MOMS heuristic, which selects the variable that has the Maximum Occurances in clauses of Minimum Size.

The MOMS variable selection heuristic does not have an obvious translation into the coefficient finding domain, but there is a fairly analogous notion. The MOMS heuristic chooses variables appearing often in small clauses because small clauses can have their satisfiability determined by examining just a few variables. Considering those variables first allows bounds on partial truth assignments to be refined quickly, potentially allowing pruning to occur sooner.

For coefficient search, this is similar to choosing a variable which, if removed, causes many examples to be identical on remaining inputs. It can be shown that if two examples differ in only one of the remaining inputs, then selecting that variable next will cause a reduction in the coefficient bound of one of the branches of the search. Selecting the variable that causes the most pairs to be identical on remaining inputs will generally result in a large

[2]This B&B algorithm was a precursor to the branch-and-bound algorithm presented in Chapter 2.

Table 7.2: A comparison of the average number of nodes visited during search to find the largest coefficient of the Fourier spectrum by several algorithms/techniques.

Data Set	B&B	B&B+Walk	B&B+MIE	BFS
Adult (34-48842)	2,225	1,819	186	35
Chess (37-3196)	139,986	272,396	1,794	212
German (24-1000)	8,457	8,232	414	209
Heart (16-270)	547	530	192	79
Pima (8-768)	22	22	27	9
Voting (16-435)	67	68	36	17
SPECT (22-267)	12,069	11,972	1,996	1,150
WBC1 (36-699)	1,939,855	2,185,064	1,615	556
WBC2 (33-198)	1,636,720	2,029,683	855	441
WBC3 (30-569)	7,555	7,562	158	57

change in bound. In the following, we shall refer to this heuristic as the Maximum Identical Examples (MIE) heuristic.

7.4.2 Empirical Results

To test the coefficient finding techniques, each was used to find the largest coefficient of the Fourier spectrum of several data sets [Newman et al., 1998]. Where necessary, non-Boolean input features were encoded into binary. The methods tested are the BFS algorithm, the generic B&B algorithm with random variable ordering, and variations on the B&B algorithm that use WalkCoef to find an initial bound (B&B+Walk) and use the MIE variable selection heuristic (B&B+MIE).

The results of the experiments are summarized in Tables 7.2 and 7.3, which show the average number of visited nodes and average run time, respectively, required to find the largest coefficient. The numbers in parentheses next to each data set name are the number of inputs and examples.

For the branch-and-bound approach, the results reveal that using the MIE variable selection heuristic greatly reduces search size and time, while using WalkCoef to find an initial

Table 7.3: A comparison of the average time (in seconds) required to find the largest coefficient of the Fourier spectrum by several algorithms/techniques.

Data Set	B&B	B&B+Walk	B&B+MIE	BFS
Adult (34-48842)	8.63	484.54	0.82	0.48
Chess (37-3196)	19.60	113.98	0.31	1.08
German (24-1000)	0.95	3.32	0.10	0.11
Heart (16-270)	0.02	0.15	0.01	0.01
Pima (8-768)	0.00	0.02	0.00	0.00
Voting (16-435)	0.00	0.21	0.00	0.00
SPECT (22-267)	0.59	1.07	0.18	0.21
WBC1 (36-699)	237.64	271.67	0.29	0.20
WBC2 (33-198)	38.39	48.14	0.06	0.06
WBC3 (30-569)	0.79	4.31	0.02	0.02

bound for pruning did not.[3] An analysis of the solutions returned by WalkCoef suggests that they are not good enough to be useful. It has been observed elsewhere that WalkSAT-like approaches are not effective at solving certain types of problems, such as those in which incremental steps towards a local optimum do not imply progress towards a globally optimal solution [Ginsberg and McAllester, 1994]. Coefficient search through the Fourier domain may be such a problem.

Comparing the results of the B&B+MIE and BFS approaches in Table 7.2 reveals that the best-first approach visits fewer nodes before finding a solution. This is not surprising, given its more sophisticated search technique. However, Table 7.3 reveals that this reduction in the number of visited nodes does not necessarily imply better performance, as the two methods perform nearly identically in terms of run time. Consequently, it seems that the additional computational overhead of the best-first approach offsets the benefit of visiting fewer nodes.

Although the BFS and B&B+MIE approaches are roughly equivalent in terms of run time, the B&B approach is superior in terms of memory efficiency. For the BFS algorithm,

[3]In a few cases, the average number of nodes visited increased when WalkCoef was used. This is only because the experiments were run independently, averaged over 100 different random variable orderings. If the same variable orderings were used, the number of visited nodes could not increase as a result of WalkCoef. The fact that there was an observed increase after 100 trials shows how much more search size is affected by variable ordering than by the initial bound produced by WalkCoef.

memory usage is proportional to the size of the search frontier, which in the worst case is $O(2^n)$. For the B&B algorithm, on the other hand, memory usage is proportional to the maximum depth of the search, which is $O(n)$. In these experiments, the BFS algorithm used roughly 10 times more memory on average than the B&B approaches. (In the tables, memory usage is reflected in the number of inputs for the B&B approach and in the number of visited nodes for the BFS approach.) Furthermore, although these experiments were small enough for the BFS algorithm to keep the search frontier in memory, the BFS algorithm will reach practical limits long before the B&B algorithm.

7.5 Conclusion

In this paper, a previously observed connection between satisfiability and spectral learning has been explored. A previous coefficient search algorithm was easily modified so that it could be applied to SAT and MAX-SAT problems, and it performs well, generally needing to explore only a small fraction of the space of possible truth assignments before arriving at a solution. Techniques for solving SAT problems have also been applied to coefficient search problems, resulting in both positive and negative results. A SAT-inspired search algorithm was presented that can find large coefficients in about the same amount of time as an existing coefficient search algorithm, while requiring much less memory. On the other hand, a SAT-inspired two-phase search approach was not effective.

In terms of the practical benefits of applying SAT algorithms to spectral learning, although we have obtained positive results, it will be interesting to see if other SAT techniques, including those not based on a recursive branch-and-bound search, may yield better results. Furthermore, we have limited consideration to exact algorithms. In some cases, finding a coefficient that is "almost large" might be sufficient for acceptable learning performance. With regards to the failure of the two-phase approach, future work will analyze the deficiencies of WalkCoef more closely and determine whether an alternative to WalkCoef would make a two-phase approach as effective in coefficient search as it has been in satisfiability.

Finally, although we have focused on applying practical techniques, it may be interesting to consider transfer of theoretical results. For example, significant work has been done in determining which classes of functions are easy/hard to learn via spectral methods and which satisfiability problems are easy/hard to solve. It would be interesting to see if there is a relationship between these results.

Chapter 8

Conclusion

Although spectral learning algorithms based on the Fourier transform have been very successful in computational learning theory and have been successfully applied in several real-world applications, previous work has left many questions and issues in spectral learning unanswered and unresolved. This dissertation addresses many of these questions and issues, making significant advances in the practical application of spectral learning methods.

Chapter 2 introduces a method for efficiently bounding the size of the largest possible coefficient in any region of a spectral representation. This method can be incorporated into a variety of search algorithms, both complete and incomplete, to allow spectral learners to find large coefficients without computing the entire exponentially-large spectrum. Chapter 2 showed that the coefficient search problem is NP-complete for a class of spectral representations that includes each of the representations considered in this dissertation. In spite of this negative result, however, empirical results with real problems show that the algorithms developed in this chapter can find the largest coefficients quickly, as they typically need to explore only a small fraction of the search space. These algorithms allow spectral learning methods to be applied to much larger problems than was possible with previous methods.

Chapter 3 analyzes the two main components of the spectral learning process (selecting the basis functions that will be used and assigning coefficients to those basis functions) and provides an empirical comparison of the most common approaches. Interestingly, empirical results suggest that the most commonly used approach, attempting to approximate the unknown function's spectral representation by selecting the basis functions whose coefficients

are largest and setting the coefficients to the values estimated from the data, is actually the least effective approach. In contrast, a boosting approach in which basis functions are selected and coefficients are computed in conjunction with a boosting algorithm is consistently more effective, while a low-order/least-squares approach that favors low-order basis functions and sets the coefficients to minimum-squared-error values is usually even better.

Chapter 4 shows how a spectral learner's performance can be improved by allowing it to use multiple spectral representations. Of three different approaches to multi-spectrum learning that are introduced and compared, an ensemble approach, in which k spectral models are learned independently (each using one of k different representations) and then combined to make classifications by majority vote, performs best. Empirical results show that a spectral learner that uses the ensemble approach can usually perform as well as or better than a single-spectrum learner that uses the best individual representation for a problem. In contrast, a single-spectrum learner that attempts to identify and use the best individual representation often fails and falls short of the best single-spectrum result.

Chapters 5 introduces the "sentiment regression" problem of learning to assign a real-valued score to a document that indicates the overall positive or negative sentiment of the author, and Chapters 6 shows how a spectral learning approach to the problem results in significant improvements over previous approaches. The common approach to solving these types of sentiment problems is to identify a set of sentiment words and then to use machine learning to build a model for classifying sentiment based on the presence/absence of those words. Chapter 6 shows how this word presence model can be extended and improved by spectral learning. It introduces a new spectral learning algorithm that combines the low-order, boosting, and multi-spectrum learning approaches. This spectral algorithm can be used to build an effective spectral model for sentiment analysis, but it can also be used as a feature selector to provide features to other algorithms. In contrast to previous failures to enhance the word presence model, the spectral feature extension of the word presence model is shown to provide significant improvements in performance.

Finally, based on the observed connection between the maximum satisfiability (MAX-SAT) problem and the problem of finding large coefficients, Chapter 7 shows how algorithms and heuristics for solving MAX-SAT problems can be adapted into effective algorithms for finding large coefficients, and vice versa. More importantly, the ease with which the algorithms from each area can be applied to the other suggests that there may be other useful practical methods and theoretical results from each area that may benefit the other.

Although the research presented in this dissertation has resulted in significant advances in spectral learning, there are still many areas for future work. For example, the analysis in Chapter 2 discusses some properties of learning problems that affect the performance of the search algorithms. However, the results on randomly generated problems having similar characteristics (based on these properties) tend to be harder on average than similarly-sized real-world problems. Therefore, one area of future work is to further identify properties of learning problems that affect performance, so there can be a better understanding of when it will or will not be easy to find large coefficients in real-world settings.

Another interesting direction for future work is to more thoroughly analyze the trade-offs and benefits of complete and incomplete approaches to finding large coefficients. For example, Chapter 2 shows that the beam search algorithm can use a very narrow beam and still find a basis function that is comparable, in terms of test accuracy, to the basis function with largest coefficient. However, how do learning accuracies compare after several basis functions are added? This comparison may be particularly interesting in the context of boosting, as incomplete algorithms may be able to be very greedy in that scenario and still match the performance obtained by a complete search.

In terms of learning spectral representations, an interesting area for future work is to identify other representations that a learner should consider. The representations used in this dissertation are effective both individually and when combined, but there are probably other representations that would also be useful. In terms of learning from multiple representations, the question of which representations work well in combination also remains open. And, a

potentially interesting direction for future work is to determine if a learner can effectively generate bases on-the-fly to adapt to specific learning problems.

Finally, the sentiment analysis research demonstrates a useful application area for spectral learning. Another important direction for future research is to identify other domains where spectral learning, and even specific spectral representations, may be useful.

References

T. Alsinet, F. Manyà, and J. Planes. Improved Branch and Bound Algorithms for MAX-SAT. In *Proceedings of the 6th International Conference on the Theory and Applications of Satisfiability Testing*, 2003.

D. Bikel and J. Sorensen. If We Want Your Opinion. In *Proceedings of the International Conference on Semantic Computing*, pages 493–500, 2007.

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 ACM Symposium on Theory of Computing*, 1994.

B. Borchers and J. Furman. A Two-Phase Exact Algorithm for MAX-SAT and Weighted MAX-SAT Problems. *Journal of Combinatorial Optimization*, 2:299–306, 1998.

N. Bshouty and C. Tamon. On the Fourier Spectrum of Monotone Functions. *Journal of the ACM*, 1996.

P. Cheeseman, B. Kanefsky, and W. M. Taylor. Where the Really Hard Problems Are. In *Proceedings of the International Joint Conference on Artificial Intelligence-91*, pages 331–337, 1991.

K. Dave, S. Lawrence, and D. Pennock. Mining the Peanut Gallery: Opinion Extraction and Semantic Classification of Product Reviews. In *Proceedings of the 12th International World Wide Web Conference*, pages 519–528, 2003.

M. Davis, G. Logemann, and D. Loveland. A Machine Program for Theorem-Proving. *Communications of the ACM*, 1962.

M. Davis and H. Putnam. A Computing Procedure for Quantification Theory. *Journal of the ACM*, 1960.

D. Donoho and I. Johnstone. Ideal Spatial Adaptation by Wavelet Shrinkage. *Biometrika*, 1994.

D. Donoho and I. Johnstone. Adapting to Unknown Smoothness via Wavelet Shrinkage. *Journal of the American Statistical Association*, 1995.

A. Drake, E. Ringger, and D. Ventura. Sentiment Regression: Using Real-Valued Scores to Summarize Overall Document Sentiment. In *Proceedings of the IEEE International Conference on Semantic Computing*, 2008.

A. Drake and D. Ventura. A Practical Generalization of Fourier-Based Learning. In *Proceedings of the International Conference on Machine Learning*, 2005.

A. Drake and D. Ventura. Search Techniques for Fourier-Based Learning. In *Proceedings of the AAAI Workshop on Search in Artificial Intelligence and Robotics*, 2008.

A. Drake and D. Ventura. Search Techniques for Fourier-Based Learning. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1040–1045, 2009.

A. Drake and D. Ventura. An Empirical Comparison of Spectral Learning Methods for Classification. 2011a. (In submission).

A. Drake and D. Ventura. Improving Spectral Learning by Using Multiple Representations. 2011b. (In submission).

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.

M. Garey and D. Johnson. *Computers and Intractability: A guide to the theory of NP-completeness*. W.H. Freeman and Co., 1979.

M. Ginsberg and D. McAllester. GSAT and Dynamic Backtracking. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, pages 226–237, 1994.

H. Hoos and T. Stützle. SATLIB: An Online Resource for Research on SAT. In *Proceedings of the International Conference on Theory and Applications of Satisfiability Testing (SAT2000)*, pages 283–292, 2000.

M. Hu and B. Liu. Mining and Summarizing Customer Reviews. In *Proceedings of the tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 168–177, 2004.

J. Jackson. An Efficient Membership-Query Algorithm for Learning DNF with Respect to the Uniform Distribution. *Journal of Computer and System Sciences*, 1997.

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.

Y. Jiang, H. Kautz, and B. Selman. Solving Problems with Hard and Soft Constraints Using a Stochastic Algorithm for MAX-SAT. In *Proceedings of the 1st International Joint Workshop on Artificial Intelligence and Operations Research*, 1995.

T. Joachims. Making Large-Scale SVM Learning Practical. In *Advances in Kernel Methods - Support Vector Learning*, pages 168–177. MIT-Press, 1999.

H. Kargupta and B. Park. Mining Decision Trees from Data Streams in a Mobile Environment. In *Proceedings of the IEEE International Conference on Data Mining*. 2001.

H. Kargupta and B. Park. A Fourier Spectrum-Based Approach to Represent Decision Trees for Mining Data Streams in Mobile Environments. *IEEE Transactions on Knowledge and Data Engineering*, 2004.

H. Kargupta, B. Park, D. Hershbereger, and E. Johnson. Collective data mining: A New Perspective Toward Distributed Data Mining. In *Advances in Distributed Data Mining*. AAAI/MIT Press, 1999.

H. Kargupta, B. Park, D Hershberger, and E. Johnson. Collective Data Mining: A New Perspective Toward Distributed Data Mining. In *Advances in Distributed and Parallel Knowledge Discovery*. AAAI/MIT Press, 2000.

H. Kargupta, B. Park, S. Pittie, L. Liu, D. Kushraj, and K. Sarkar. MobiMine: Monitoring the Stock Market from a PDA. In *ACM SIGKDD Explorations Newsletter*, 2002.

A. Klivans, R. O'Donnell, and R. Servedio. Learning Intersections and Thresholds of Halfspaces. *Journal of Computer and System Sciences*, 68:808–840, 2004.

E. Kushilevitz and Y. Mansour. Learning Decision Trees using the Fourier Spectrum. *SIAM Journal on Computing*, 1993.

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.

N. Linial, Y. Mansour, and N. Nisan. Constant Depth Circuits, Fourier Transform, and Learnability. *Journal of the ACM*, 1993.

B. Liu, M. Hu, and J. Cheng. Opinion Observer: Analyzing and Comparing Opinions on the Web. In *International World Wide Web Conference*, 2005.

Y. Mansour. Learning Boolean Functions via the Fourier Transform. In V.P. Roychodhury, K-Y. Siu, and A. Orlitsky, editors, *Theoretical Advances in Neural Computation and Learning*, pages 391–424. Kluwer Academic Publishing, 1994.

Y. Mansour. An $O(n^{\log\log n})$ Learning Algorithm for DNF under the Uniform Distribution. *Journal of Computer and System Sciences*, 1995.

Y. Mansour and S. Sahar. Implementation Issues in the Fourier Transform Algorithm. *Machine Learning*, 2000.

T. Nasukawa and J. Yi. Sentiment Analysis: Capturing Favorability using Natural Language Processing. In *Proceedings of the International Conference On Knowledge Capture*, 2003.

D.J. Newman, S. Hettich, C.L. Blake, and C.J. Merz. UCI Repository of Machine Learning Databases, 1998. URL `http://www.ics.uci.edu/$\sim$mlearn/MLRepository.html`.

B. Pang and L. Lee. A Sentimental Education: Sentiment Analysis using Subjectivity Summarization Based on Minimum Cuts. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 271–278, 2004.

B. Pang and L. Lee. Seeing Stars: Exploiting Class Relationships for Sentiment Categorization with Respect to Rating Scales. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 115–124, 2005.

B. Pang, L. Lee, and S. Vaithyanathan. Thumbs Up? Sentiment Classification using Machine Learning Techniques. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 79–86, 2002.

B. Park, R. Aggayari, and H. Kargupta. A Fourier Analysis Based Approach to Learning Decision Trees in a Distributed Environment. In *Proceedings of the SIAM International Conference on Data Mining*, 2001.

A. Popescu and O. Etzioni. Extracting Product Features and Opinions from Reviews. In *Proceedings of the Human Language Technology Conference and the Conference on Empirical Methods in Natural Language Processing*, pages 339–346, 2005.

P. Turney. Thumbs Up or Thumbs Down? Semantic Orientation Applied to Unsupervised Classification of Reviews. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 417–424, 2002.

T. Wilson, J. Wiebe, and P. Hoffmann. Just How Mad Are You? Finding Strong and Weak Opinion Clauses. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI '04)*, pages 761–769, 2004.

T. Wilson, J. Wiebe, and P. Hoffmann. Recognizing Contextual Polarity in Phrase-Level Sentiment Analysis. In *Proceedings of the Human Language Technology Conference and the Conference on Empirical Methods in Natural Language Processing*, pages 347–354, 2005.

H. Yu and V. Hatzivassiloglou. Towards Answering Opinion Questions: Separating Facts from Opinions and Identifying the Polarity of Opinion Sentences. In *Proceedings of the 2003 Conference on Empirical Methods in Natural Language Processing*, pages 129–136, 2003.