A Practical Generalization of Fourier-based Learning

Adam Drake ACD2@CS.BYU.EDU
Dan Ventura VENTURA@CS.BYU.EDU
Computer Science Department, Brigham Young University, Provo, UT 84602 USA

Abstract

This paper presents a search algorithm for finding functions that are highly correlated with an arbitrary set of data. The functions found by the search can be used to approximate the unknown function that generated the data. A special case of this approach is a method for learning Fourier representations. Empirical results demonstrate that on typical real-world problems the most highly correlated functions can be found very quickly, while combinations of these functions provide good approximations of the unknown function.

1. Introduction

The discrete Fourier transform converts a function into a unique spectral representation in which it is represented as a linear combination of Fourier basis functions. The ability to represent functions as a combination of basis functions led to the development of learning algorithms based on the Fourier transform. These Fourier-based learning algorithms, which have been used primarily in the field of computational learning theory, learn functions by approximating the coefficients of the most highly correlated basis functions.

The first Fourier-based learning algorithm was introduced by Linial, Mansour, and Nisan (1993). They presented an algorithm (hereafter referred to as the LMN algorithm) that learns functions by approximating the coefficients of the low-order basis functions. Given sufficient training examples drawn from a uniform distribution, the LMN algorithm can effectively learn any function whose spectral representation is concentrated on the low-order coefficients.

Another important Fourier-based learning algorithm was introduced by Kushilevitz and Mansour (1993). Their algorithm (hereafter referred to as the KM algorithm) does not require a function's spectral representation to be concentrated on the low-order coefficients. Instead, it recursively searches the space of Fourier basis functions to find, with high probability, the basis functions with the largest coefficients. It relies on a membership oracle (a black box that can be queried to learn the value of the function at any point) to efficiently carry out its search.

These algorithms have been successfully used to prove learnability results for many classes of problems. However, there has been little work in applying Fourier-based algorithms to real-world problems. The primary difficulty in applying Fourier techniques to real-world problems is that the number of Fourier basis functions, and the time required to compute the Fourier transform, is exponential in the number of inputs to a function. The LMN and KM algorithms avoid exponential complexity by imposing restrictions that limit real-world applicability.

The LMN algorithm avoids exponential complexity by approximating only the low-order coefficients. This restriction is undesirable because it limits the set of functions that can effectively learned. Furthermore, there is generally nothing known about the spectral representation of a real-world learning problem, making it impossible to know beforehand whether the restriction to low-order coefficients is acceptable.

The KM algorithm avoids exponential complexity by relying on a membership oracle to guide its search for large coefficients. The requirement of a membership oracle greatly limits the applicability of the algorithm. Mansour and Sahar (2000) presented results of effectively applying the KM algorithm to a real-world problem for which a membership oracle exists. Unfortunately, however, the existence of a membership oracle is not typical of many learning scenarios.

The main result of this paper, which may be useful beyond the field of machine learning, is a new search

Appearing in *Proceedings of the 22^{nd} International Conference on Machine Learning*, Bonn, Germany, 2005. Copyright 2005 by the author(s)/owner(s).

algorithm that efficiently searches a space of functions to find those that are most highly correlated with an arbitrary set of data. A special case of this algorithm is a method for finding the most highly correlated Fourier basis functions. This search allows Fourier-based learning algorithms to overcome the limitations of previous approaches.

In addition to describing the search algorithm and demonstrating its effectiveness, results of learning real-world problems with the results of the search are presented. The first approach is a standard Fourier-based approach of learning a linear combination of Fourier basis functions. However, because the search algorithm can be used to find other highly correlated functions, the basic Fourier approach is generalized to allow other types of functions to be used in the linear combination. This extension, which is a departure from Fourier theory, has been found to be very useful in practice.

2. Definitions and Notation

Previous work in Fourier-based learning has been concerned with Boolean functions and the Fourier transform for Boolean functions, which is also known as a Walsh transform. The algorithms and results presented in this paper are also for Boolean functions, although many of the ideas presented could be applied to discrete-valued functions and the more general discrete Fourier transform.

Let f be a Boolean function of the form $f : \{0,1\}^n \rightarrow \{1,-1\}$ where n is the number of inputs. There are 2^n Fourier basis functions for a function of n Boolean inputs, each indexed by a binary number $\alpha \in \{0,1\}^n$. Each basis function χ_α is defined as follows:

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

where $x \in \{0,1\}^n$. Note that each basis function computes the parity (or logical XOR) of a subset of the inputs. Specifically, basis function χ_α computes the parity of those inputs x_i for which $\alpha_i = 1$. There is one basis function for each possible subset of inputs.

The Fourier coefficients $\hat{f}(\alpha)$ are defined by:

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

Thus, each coefficient is computed by taking the dot product of the outputs of functions f and χ_α and therefore measures the correlation between f and χ_α. In other words, basis functions that are highly correlated with f have coefficients with large absolute values. (A negative coefficient indicates an inverse correlation.)

Given the Fourier basis functions and coefficients as defined above, any Boolean function f can be represented as a linear combination of the basis functions:

$$f(x) = \sum_{\alpha=0}^{2^n-1} \hat{f}(\alpha)\chi_\alpha(x) \tag{3}$$

A Fourier-based learning algorithm can approximate the Fourier coefficients when the function is only partially known, as is the case in typical learning scenarios. Let X be a set of available training data, with x being a particular training example. Then the Fourier coefficients can be approximated by the following:

$$\tilde{\hat{f}}(\alpha) = \frac{1}{|X|} \sum_{x \in X} f(x)\chi_\alpha(x) \tag{4}$$

As described previously, the search algorithm presented in this paper can find functions other than the Fourier basis functions. In particular, two additional types of Boolean functions are considered: functions that compute the conjunction (logical AND) of subsets of the inputs and functions that compute the disjunction (logical OR) of subsets of the inputs. These functions can be defined in a manner similar to the Fourier basis functions as follows:

$$AND_\alpha(x) = \begin{cases} +1 : \text{if } \sum_{i=0}^{n-1} x_i\alpha_i = \sum_{i=0}^{n-1} \alpha_i \\ -1 : \text{if } \sum_{i=0}^{n-1} x_i\alpha_i < \sum_{i=0}^{n-1} \alpha_i \end{cases} \tag{5}$$

$$OR_\alpha(x) = \begin{cases} +1 : \text{if } \sum_{i=0}^{n-1} x_i\alpha_i > 0 \\ -1 : \text{if } \sum_{i=0}^{n-1} x_i\alpha_i = 0 \end{cases} \tag{6}$$

Coefficients for the AND and OR functions can be computed in the same manner as shown in (4) simply by replacing χ_α with either AND_α or OR_α. These "coefficients" measure the correlation with the function f, but otherwise do not have the same meaning as the Fourier basis function coefficients. Neither the AND nor the OR functions form a basis for the space of Boolean functions, and the coefficients do not generally yield a linear combination for representing f.

As each AND, OR, and parity function measures the correlation between itself and the training data, we refer to these functions as correlation functions.

3. A Best-first Search for Correlation Functions

This section presents the algorithm for finding correlation functions that are highly correlated with an arbitrary set of data. Empirical results show that although the search space is exponential in size, it is possible to find solutions while exploring only a fraction of the space.

3.1. The Algorithm

The algorithm uses a best-first search to explore the possible subsets of inputs over which a particular correlation function could be defined. The search space can be represented as a binary tree of nodes, n levels deep. Each node represents a partially or completely specified correlation function label. At the top of the tree is a node with a completely unspecified label. At each successive level of the tree one additional digit of the label is set, until finally at the leaf nodes the labels are completely specified. There are 2^n leaf nodes, one for each possible correlation function.

A best-first search is beneficial because as each digit of a correlation function label is set, some information can be obtained about the maximum coefficient of any correlation function whose label shares that digit value. By exploring nodes that could potentially lead to higher coefficients first, it is possible to find the most highly correlated functions while exploring only a fraction of the search space.

Consider the information gained when digit 0 of an AND function's label is set to 1. All AND functions for which $\alpha_0 = 1$ include input x_0 in their calculation. Thus, any training examples for which $x_0 = 0$ will be classified as false. If there are two examples, x and x', such that $x_0 = x'_0 = 0$ but $f(x) \neq f(x')$ then no AND_α such that $\alpha_0 = 1$ will be able to correctly classify both examples. In fact, they will correctly classify one and only one of them, regardless of what the other digits in the label are. Each conflicting pair of examples reduces the maximum possible coefficient value of functions for which $\alpha_0 = 1$ by 2. (Technically, the coefficient value is reduced by $2/|X|$, but for simplicity the normalization can be ignored during search.)

In describing the algorithm more precisely, some additional notation will be useful. As before, let X be the set of training examples, or known points of the target function f, with each training example being a pair of the form $(x, f(x))$, where $x \in \{0,1\}^n$ and $f(x) \in \{-1,1\}$. Let $\alpha \in \{0,1\}^n$ be the label of some correlation function C, and let C_α be the correlation function with label α. As described in section 2, the digits of α indicate the subset of inputs over which C_α is defined. Let $\beta \in \{0,1,*\}^n$ be a correlation function label in which some digits may be unspecified, with each $*$ indicating an unspecified digit. We shall say that $\alpha \subseteq \beta$ if for $0 \leq i < n$, $\alpha_i = \beta_i$ or $\beta_i = *$. Finally, let $\hat{f}_{max}(\beta)$ be the maximum coefficient of any correlation function C_α such that $\alpha \subseteq \beta$.

The search algorithm is efficiently implemented with a priority queue of nodes, which sorts nodes in decreasing order of $\hat{f}_{max}(\beta)$. Each node contains a correlation function label β, a set of training examples X_β, and a variable $count_\beta$. The root node of the search is initialized with a completely unspecified correlation function ($\beta = *^n$), all of the training examples ($X_\beta = X$), and $count_\beta = 0$.

The algorithm begins by inserting the root node into a priority queue. Then, it enters a loop that continues until the desired number of correlation functions has been found. Each time through the loop, the node at the front of the queue is removed. If the label β of that node is completely specified, then β is added to the set of highly correlated functions. (Because the node with the highest value for $\hat{f}_{max}(\beta)$ is always at the front of the priority queue, a solution node at the front of the queue is certain to be at least as good as any other potential solution.) If β is not completely specified, then one of the unspecified digits is selected, and the two child nodes that result from setting that digit to 0 and 1 are created and inserted into the queue.

The value of $\hat{f}_{max}(\beta)$ can be computed for any internal node as follows:

$$\hat{f}_{max}(\beta) = abs(count_\beta) + |X_\beta|$$

where $abs(count_\beta)$ is the absolute value of $count_\beta$. Each time a child node is created and given a label β, the examples of the parent node are checked to see if any pairs of examples cannot be both correctly classified by any C_α such that $\alpha \subseteq \beta$. Any such pairs of examples are removed. Because the unnormalized coefficient of a correlation function is reduced by two for each pair of examples such that one and only one of them can be classified correctly, the number of examples removed in this way is equal to the reduction in $\hat{f}_{max}(\beta)$. When finding Fourier basis (XOR) functions, $count_\beta$ is always zero. Its use when finding AND and OR functions is described later.

The value of $\hat{f}_{max}(\beta)$ is computed for leaf nodes by the following:

$$\hat{f}_{max}(\beta) = abs(count_\beta \pm |X_\beta|)$$

where $|X_\beta|$ is subtracted from $count_\beta$ if X_β contains positive examples, and added to $count_\beta$ if X_β contains negative examples. (At leaf nodes, there will never be both positive and negative examples in X_β.) The value of $\hat{f}_{max}(\beta)$ at a leaf node is equivalent to the absolute value of $\hat{f}(\beta)$.

```
FindCorrelationFunctions($X$, $numFunctionsToFind$)
{
    initialize a node with $\beta = *^n$ and $X_\beta = X$
    insert initial node into priority queue
    $M \leftarrow \{\}$

    while $|M| < numFunctionsToFind$
    {
        remove the node at the front of the queue

        if for that node $\beta_i \neq *$ for $0 \leq i < n$, $M \leftarrow M \cup \beta$
        else
        {
            determine which digit of $\beta$ to set next
            create the 0 and 1 child nodes
            insert the nodes into the priority queue
        }
    }

    return $M$
}
```

Figure 1. General algorithm for finding the functions that are most highly correlated with an arbitrary set of data.

The order in which digits of β are set is chosen with the intent of reducing the number of nodes that must be visited. The method used in our current implementation is to select the digit that minimizes the following:

$$\arg\min_i(\min(\hat{f}_{max}(\beta_i = 0), \hat{f}_{max}(\beta_i = 1))$$

in which $\hat{f}_{max}(\beta_i = 0)$ and $\hat{f}_{max}(\beta_i = 1)$ indicate the values of $\hat{f}_{max}(\beta)$ for the child nodes of the current node. This heuristic selects the digit that causes the greatest reduction in $\hat{f}_{max}(\beta)$ for either child.

The intuition behind this heuristic is that larger decreases in $\hat{f}_{max}(\beta)$ indicate more information gain. Consider the worst case, in which $\hat{f}_{max}(\beta)$ does not decrease for either child. In this case, the algorithm has been given no useful information regarding the maximum coefficient down each path.

The algorithm for finding correlation functions is shown in Figure 1. The only portion of the algorithm that is significantly different for each type of correlation function is the step that creates the 1 child node (the node that is created by setting the selected digit to 1). For each type of correlation function there is a unique way of creating this child node. The methods for creating the 0 and 1 child nodes are depicted in Figures 2 and 3.

When the 0 child node is created, the process of determining which examples to remove from the child is fairly straightforward. Pairs of examples are removed if the examples' inputs differ in only those digits of β that have previously been specified and if they have different outputs. The reason for this is that if $\beta_i = 0$, then any examples that differ only in the i^{th} input should not have different outputs, because that input will be ignored. If a pair of such examples do have different outputs, all C_α such that $\alpha \subseteq \beta$ will classify one and only one of them correctly.

```
CreateZeroChild(Node $parent$, int $i$)
{
    Node $child$
    $child.\beta \leftarrow parent.\beta$
    $child.\beta_i \leftarrow 0$
    $child.X_\beta \leftarrow parent.X_\beta$
    $child.count_\beta \leftarrow parent.count_\beta$

    for each $x, x' \in child.X_\beta$
        if $f(x) \neq f(x')$ and $\forall j$ $(\beta_j \neq * \vee x_j = x'_j)$
            remove $x$ and $x'$ from $child.X_\beta$

    return $child$
}
```

Figure 2. Method for creating the "0" child of a node. i is the digit of β to be set in the child.

The method for determining which examples should not be added to the 1 child is identical to that of the 0 child, except that some preprocessing must be done to the examples. For nodes representing parity functions, the preprocessing involves inverting the output of all examples such that $x_i = 1$, where i is the digit currently being set in the label. The reason for this step is that if $\beta_i = 1$ then $\chi_\beta(x)$ will be inverted for each example x such that $x_i = 1$, while its output will remain the same if $x_i = 0$. By inverting the output of each x such that $x_i = 1$, the same example removing method used to create the 0 child can be used, while correctly accounting for how the parity functions classify the examples.

The preprocessing steps required for AND and OR functions are quite similar to each other. For AND functions, each example x for which $x_i = 0$ is removed, and its output is added to $count_\beta$. These examples are removed because as soon as there is an i such that $\beta_i = 1$ and $x_i = 0$ it is certain that $AND_\alpha(x) = -1$ for all $\alpha \subseteq \beta$. By adding $f(x)$ to $count_\beta$, $count_\beta$ keeps a running total of how many examples whose classifications are already determined can still be classified correctly. Note that a positive and negative example will cancel each other out, which has the same effect on $\hat{f}_{max}(\beta)$ as removing two examples from X_β. OR functions are treated in the same way as AND func-

```
CreateOneChild(Node $parent$, int $i$)
{
    Node $child$
    $child.\beta \leftarrow parent.\beta$
    $child.\beta_i \leftarrow 1$
    $child.X_\beta \leftarrow parent.X_\beta$
    $child.count_\beta \leftarrow parent.count_\beta$

    if $parent$ represents a parity function
    {
        for each $x \in child.X_\beta$ such that $x_i = 1$
            $f(x) \leftarrow -f(x)$
    }
    if $parent$ represents an AND function
    {
        for each $x \in child.X_\beta$ such that $x_i = 0$
            $child.count_\beta \leftarrow child.count_\beta + f(x)$
            remove $x$ from $child.X_\beta$
    }
    if $parent$ represents an OR function
    {
        for each $x \in child.X_\beta$ such that $x_i = 1$
            $child.count_\beta \leftarrow child.count_\beta + f(x)$
            remove $x$ from $child.X_\beta$
    }

    for each $x, x' \in child.X_\beta$
        if $f(x) \neq f(x')$ and $\forall j \ (\beta_j \neq * \vee x_j = x'_j)$
            remove $x$ and $x'$ from $child.X_\beta$

    return $child$
}
```

Figure 3. Method for creating the "1" child of a node. i is the digit of β to be set in the child.

tions, except that examples are removed if $x_i = 1$, which indicates that $OR_\alpha(x) = 1$ for all $\alpha \subseteq \beta$.

The algorithm can be made to search for multiple types of correlation functions simultaneously simply by initializing the priority queue with one node for each type of correlation function. When finding multiple types of correlations, the algorithm can be thought of as searching multiple search trees simultaneously, switching between trees to explore the path that currently looks most promising. Some means of determining the type of correlation function a node represents must be added so that nodes can be correctly handled by the algorithm (for example, by adding a *type* field to each node).

The following theorem makes the claim for the correctness of the algorithm.

Theorem 1. *The best-first search algorithm described in Figure 1 returns the labels of the correlation functions that are most highly correlated with X.*

Proof Sketch. The proof of the theorem follows from two facts. First, the structure of the search space ensures that all possible correlation functions are considered. And second, because the node with the highest value of $\hat{f}_{max}(\beta)$ is always at the front of the priority queue, correlation functions are always added to the solution set in order of highest coefficient value. (This assumes that $\hat{f}_{max}(\beta)$ is correctly computed for each node.)

The time complexity of the algorithm is $O(mnk \log k)$, where n is the number of inputs, k is the number of training examples, and m is the number of nodes visited during the search. The $nk \log k$ term indicates the time required to create the child nodes of the current node, and assumes that training examples are stored in a structure that allows lookups in $\log k$ time. The number of nodes visited, m, may vary from n to $2^n - 1$. Thus, the algorithm is ultimately bounded by an exponential term, and relies on the assumption that in practice m will not approach that bound. Experiments on real-world problems suggest that typical searches explore only a fraction of the search space.

3.2. Empirical Evaluation

To test the algorithm, nine data sets were selected from the UCI machine learning repository (1998). Each data set represents a Boolean classification problem, although several of the sets contain non-Boolean input features, which were encoded into binary as follows. Real-valued inputs were encoded as a single binary value, indicating whether the value was above or below some threshold. (The threshold was chosen by sorting examples in increasing order of that attribute value, considering all thresholds that lie at the midpoints between the values of adjacent examples, and selecting the threshold that best separated the examples into positive and negative classes.) Nominal-valued inputs were encoded into binary using the minimum number of bits needed to encode all possible values.

Table 1 shows the number of nodes expanded to find the 1 and 1,000 parity functions that are most highly correlated with each data set, compared to the total number of possible node expansions. (Note that the Pima data set, a problem with 8 inputs, has only 256 basis functions, and therefore has no data for finding 1,000 functions.) As can be seen, only a small fraction of the search space needs to be explored to find a large number of parity functions.

The run time required by the algorithm was also very small. Table 2 shows the time required by the algorithm to find those same numbers of functions, compared to the time required to compute the fast Walsh

Table 1. Nodes visited to find the 1 and 1,000 most highly correlated parity functions, compared to the total number of possible nodes. The number of attributes for each data set is shown in parentheses.

Data set	1	1,000	Total possible
Chess (37)	209	2,010	137,438,953,472
German (24)	197	3,378	16,777,215
Heart (16)	73	2,804	65,535
Pima (8)	8	n/a	256
SPECT (22)	1,115	114,868	4,194,303
Voting (16)	16	2,514	65,535
Wisc 1 (36)	540	126,474	68,719,476,735
Wisc 2 (33)	322	3,621	8,589,934,592
Wisc 3 (30)	44	2,445	1,073,741,824

Table 2. Time required to find the 1 and 1,000 most highly correlated parity functions, compared to the time required to compute the fast Walsh transform. The number of attributes for each data set is shown in parentheses.

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

transform (FWT). The FWT, which computes the entire spectrum in $O(n2^n)$ time, is a Boolean version of the fast Fourier transform algorithm. Although the FWT can quickly calculate the Fourier spectra of problems for which n is small, its run time increases dramatically as n increases. (Note that for the Chess, Wisc 1, and Wisc 2 datasets the standard FWT algorithm requires more memory than can be allocated and addressed in a typical 32-bit workstation. The run times shown are estimates based on observing the growth in run time as n increases.)

Results for finding highly correlated AND and OR functions are generally even better. This is largely due to the fact that the output of an AND or OR function can often be determined by examining a single input, while the output of a parity function cannot be determined until all x_i such that $\beta_i = 1$ have been checked.

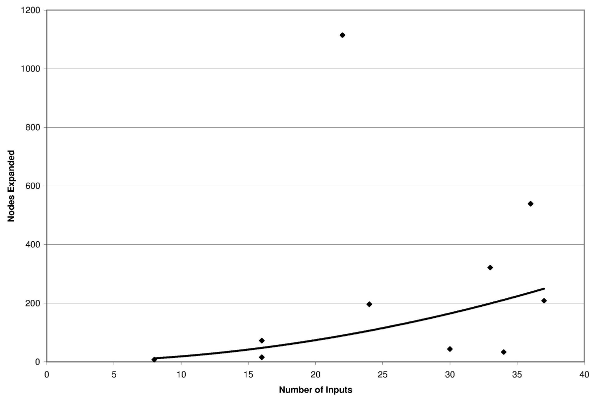

Figure 4. A plot of the number of nodes expanded to find the most highly correlated Fourier basis function vs. the number of inputs. The trendline is a least squares fit of a single-term polynomial to the data.

3.3. Analysis

Although the search algorithm has a worst-case exponential time complexity, it performs well on experiments with real-world data. Like other searches of this type its expected real-world performance is difficult to characterize meaningfully because m, the number of nodes expanded, is very problem dependent. However, m is loosely related to n, the number of inputs, and k, the number of training examples.

The value of n certainly increases the upper bound on m, but empirical results suggest that in practice the value of m is not exponential in n. A least squares fit of a trendline to the plot of m vs. n for these data sets reveals that a single-term polynomial equation fits the data better than an exponential equation. Figure 4 shows a plot of m vs. n for the case of finding a single parity function, with the best-fit single-term polynomial equation included. The equation for this trendline has an exponent of 1.9, suggesting that $m \propto n^2$. Empirical results indicate that as the number of correlation functions found increases, the exponent decreases. For example, when finding 1,000 parity functions, the exponent is 0.3, indicating a sub-linear relationship.

The value of k has a somewhat non-intuitive effect on the algorithm. Although the time required to expand a node increases with k, the number of nodes expanded generally decreases with k. This is because more examples allow the algorithm to distinguish between correlation functions more quickly. Therefore, despite the fact that k factors into the time complexity of the algorithm, increases in k can decrease the run-time of the algorithm.

4. Learning with Correlation Functions

The best-first search algorithm presented in Section 3 was developed to make it possible to implement Fourier-based learning algorithms that are practical for solving real-world problems, without sacrificing representational power or applicability. The results of the previous section demonstrate that it is possible to quickly find the functions that are most highly correlated with real-world data. This section presents results of learning real-world problems using the best-first search for correlation functions.

4.1. Learning method

In the most straightforward application of Fourier-based learning, the most highly correlated basis functions are found, weighted by their coefficients, and combined into a single classifier that uses the weighted vote of basis functions to make classifications. Let M be the set of labels of highly correlated basis functions found in the search. Given an input x, the Fourier-based classifier c makes classifications as follows:

$$c(x) = \begin{cases} +1 & : \text{ if } \sum_{\alpha \in M} \tilde{\hat{f}}(\alpha)\chi_\alpha(x) \geq 0 \\ -1 & : \text{ if } \sum_{\alpha \in M} \tilde{\hat{f}}(\alpha)\chi_\alpha(x) < 0 \end{cases} \tag{7}$$

Although this method works well, we have found that results can often be improved by altering the coefficients from those given by (4). In particular, we modify the coefficients using incremental gradient descent to minimize the error of the classifier. This can be thought of as learning an optimal set of coefficients for the selected basis functions.

4.2. Empirical Evaluation

The learning accuracies achieved when testing on real-world problems have been quite favorable. Table 3 presents the accuracy of the standard Fourier-based learner (described above) on the data sets used in Section 3. Results are compared to an implementation of the C4.5 decision tree algorithm and to an enhanced Fourier-based learner (described later). Each accuracy represents the average of 30 10-fold cross validations. The standard Fourier-based learning results were obtained by finding up to 1,000 highly correlated basis functions and using gradient descent on the coefficients to reduce error. In several cases, limiting the standard Fourier-based learner to fewer than 1,000 basis functions improved generalization.

As can be seen in Table 3, the standard Fourier-based learner performs fairly well. However, the decision tree algorithm consistently performs better.

Table 3. Test accuracy of a standard Fourier-based learner (Standard), of a Fourier-based learner enhanced with AND and OR functions (Enhanced), and of C4.5. Accuracies are the average of 30 10-fold cross validations. Statistically significant improvements of the enhanced Fourier learner over the standard Fourier learner are highlighted in bold.

Data set	Standard	Enhanced	C4.5
Chess	85.9	81.0	99.4
German	71.5	71.5	73.4
Heart	79.9	**84.4**	81.5
Pima	73.6	**76.1**	74.5
SPECT	79.4	**84.0**	80.9
Voting	96.3	96.3	96.6
Wisc 1	94.6	**95.1**	94.6
Wisc 2	71.0	**74.9**	75.8
Wisc 3	91.5	**93.6**	94.4

Inspired by recent work suggesting that AND and OR functions may generally be more useful than XOR functions for solving typical real-world learning problems (Drake & Ventura, 2005), the standard Fourier approach was extended to use AND and OR functions in addition to XOR functions. For many of the data sets, this resulted in a significant improvement in accuracy. These results are shown in Table 3 under the label Enhanced. Those accuracies shown in bold indicate statistically significant improvements of the enhanced Fourier-based learner over the standard Fourier-based learner. (Statistical significance was measured with a random permutation test. In each of the statistically significant cases, 100,000 random permutations were tested, none of which yielded a larger difference in accuracy than the observed difference.)

Note that the enhanced Fourier-based learner performs much worse on the Chess data set than C4.5, and even performs worse than the standard learner. Additional experiments suggest that this may be a result of the correlation functions being too highly correlated with each other. Future work will include exploring ways of overcoming this difficulty.

4.3. Analysis

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

Another useful property of the algorithm is its ability to find several types of correlations. Some learning algorithms learn a particular type of correlation well, but struggle to learn others. For example, an algorithm may learn conjunctions of attributes quite well, but struggle to learn XOR relationships. The algorithm presented here is capable of finding and using AND, OR, and XOR correlations. In addition, if deemed useful, other types of correlations could be added to the search.

As a final note, an additional approach to Fourier-based learning that has been considered is a boosting approach in which basis functions are treated as weak hypotheses combined to form a strong hypothesis. Such an approach was introduced by Jackson (1997). We have applied the *AdaBoost* algorithm of Freund and Schapire (1996) to our Fourier-based learner, but with mixed results. The boosting algorithm successfully reduced the number of basis functions needed to correctly classify the training data, and accuracy on the Chess data set was improved to 95%. For many of the data sets, however, the boosted classifier did not generalize as well. Finding a solution to this problem will be an area of future research.

5. Additional Applications

Although the algorithm for finding correlation functions is interesting as the foundation of a learning algorithm, it may be beneficial in other areas as well. One such area is data analysis. By exploring a large space of useful correlations, interesting properties of the data can be efficiently determined. For example, running the algorithm to find correlation functions for the SPECT data set reveals an OR function with a strong correlation to the output (much stronger than any AND or XOR correlations). This particular correlation function computes the logical OR of eight inputs. Such a high-order correlation would be nearly impossible for humans to observe, but the search algorithm finds it in seconds.

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

6. Conclusion

The best-first search algorithm for finding functions that are highly correlated with an arbitrary set of data has been shown to be highly effective in practice. It allows Fourier-based learning algorithms to be efficiently applied to real-world problems without limiting the search for large coefficients or requiring a membership oracle. The search algorithm can also be used to find other types of highly correlated functions, greatly increasing its usefulness.

Areas for future work include extending the techniques presented here to non-Boolean functions, exploring the benefits of adding other types of functions to the search, determining better ways to find/combine functions for learning (including improving the mixed results of the boosting approach), and investigating methods for automating the process of determining which types of correlation functions to search for.

References

Blake, C., & Merz, C. (1998). UCI repository of machine learning databases.

Drake, A., & Ventura, D. (2005). Comparing high-order boolean features. *Proceedings of the 8th Joint Conference on Information Sciences*, to appear.

Freund, Y., & Schapire, R. (1996). Experiments with a new boosting algorithm. *Proceedings of the 13th International Conference on Machine Learning*, *55*, 148–156.

Jackson, J. (1997). An efficient membership-query algorithm for learning dnf with respect to the uniform distribution. *Journal of Computer and System Sciences*, *55*, 414–440.

Kushilevitz, E., & Mansour, Y. (1993). Learning decision trees using the fourier spectrum. *SIAM Journal on Computing*, *22*, 1331–1348.

Linial, N., Mansour, Y., & Nisan, N. (1993). Constant depth circuits, fourier transform, and learnability. *Journal of the ACM*, *40*, 607–620.

Mansour, Y., & Sahar, S. (2000). Implementation issues in the fourier transform algorithm. *Machine Learning*, *14*, 5–33.