Search Techniques for Fourier-Based Learning

Adam Drake and **Dan Ventura**
Computer Science Department
Brigham Young University
{acd2,ventura}@cs.byu.edu

Abstract

Fourier-based learning algorithms rely on being able to efficiently find the large coefficients of a function's spectral representation. In this paper, we introduce and analyze techniques for finding large coefficients. We show how a previously introduced search technique can be generalized from the Boolean case to the real-valued case, and we apply it in branch-and-bound and beam search algorithms that have significant advantages over the best-first algorithm in which the technique was originally introduced.

1 Introduction

Fourier-based learning algorithms attempt to learn a function by approximating its Fourier representation. They have been used extensively in learning theory, where properties of the Fourier transform have made it possible to prove many learnability results [Jackson, 1997; Kushilevitz and Mansour, 1993; Linial *et al.*, 1993]. Fourier-based algorithms have also been effectively applied in real-world settings [Drake and Ventura, 2005; Kargupta *et al.*, 1999; Mansour and Sahar, 2000].

In order to approximate a function's Fourier representation, a learning algorithm must be able to efficiently identify the large coefficients of the spectrum. Thus, Fourier-based learning is essentially a search problem, as the effectiveness of this learning approach is tied to the search for large coefficients.

In this paper, we consider the problem of finding large spectral coefficients in real-world settings. Specifically, we consider the scenario in which a learner is asked to learn an unknown function from a set of labeled examples. In the following sections, we briefly discuss the hardness of the search problem, and describe a coefficient search technique that can be incorporated into a variety of search algorithms. We show that a complete branch-and-bound algorithm is as fast as and more memory efficient than a previously introduced algorithm. We also introduce an incomplete beam search algorithm that is always fast and usually able to find solutions that are as good as those found by the complete algorithms.

2 Fourier-Based Learning

The algorithms discussed in this paper are based on the Fourier transform of Boolean-input functions. It is also known as a Walsh transform.

2.1 The Fourier Transform

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

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

where $\alpha \in \{0,1\}^n$ and $\hat{f}(\alpha)$ is the spectral coefficient of basis function χ_α. Each $\chi_\alpha : \{0,1\}^n \longrightarrow \{1,-1\}$ is defined by the following:

$$\chi_\alpha(x) = \left\{ \begin{array}{rl} 1 & : \text{if } \sum_i \alpha_i x_i \text{ is even} \\ -1 & : \text{if } \sum_i \alpha_i x_i \text{ is odd} \end{array} \right. \tag{2}$$

where α_i and x_i denote the ith binary digits of α and x.

The 2^n basis functions of the Fourier transform are XOR functions, each computing the XOR of a different subset of inputs. The subset is determined by α, as only inputs for which $\alpha_i = 1$ contribute to the output of χ_α. The order of a basis function is the number of inputs for which $\alpha_i = 1$.

The Fourier coefficients provide global information about f. For example, each $\hat{f}(\alpha)$ provides a measure of the correlation between f and χ_α. A large positive or negative value indicates a strong positive or negative correlation, while a small value indicates little or no correlation.

Every f can be expressed as a linear combination of the basis functions, and the Fourier coefficients provide the correct linear combination:

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

Equation 3 is the inverse transform, and it shows how any f can be recovered from its Fourier representation.

2.2 Learning Fourier Representations

In typical learning scenarios, f, the target function, is unknown and must be learned from a set X of $\langle x, f(x) \rangle$ examples of the function. A Fourier-based learning algorithm

attempts to learn a linear combination of a subset B of the basis functions that is a good approximation of f:

$$f(x) \approx \sum_{\alpha \in B} \hat{f}_X(\alpha)\chi_\alpha(x) \qquad (4)$$

Here, $\hat{f}_X(\alpha)$ denotes a coefficient approximated from X. Since f is only partially known, the true values of the coefficients cannot be known with certainty. However, they can be approximated from the set of examples:

$$\hat{f}_X(\alpha) = \frac{1}{|X|} \sum_{\langle x, f(x) \rangle \in X} f(x)\chi_\alpha(x) \qquad (5)$$

It can be shown that under certain conditions coefficients approximated in this way will not differ much from the true coefficients [Linial *et al.*, 1993].

Since there are an exponential number of basis functions (2^n basis functions for an n-input function), it is not practical to use all basis functions unless n is small. Consequently, a key difference between Fourier-based algorithms is in the choice of which basis functions will be used. (The other basis functions are implicitly assumed to have coefficients of 0.)

One approach is to use all basis functions whose order is less than or equal to some value k [Linial *et al.*, 1993]. This approach has the advantage that no search for basis functions is necessary. However, it has the disadvantage that the class of functions that can be learned is limited to those that can be expressed with only low-order basis functions.

Another approach is to search for and use any basis functions whose coefficients are "large" (e.g., larger than some threshold θ) [Kushilevitz and Mansour, 1993; Mansour and Sahar, 2000]. Basis functions with large coefficients carry most of the information about a function, and many functions can be approximated well by only the large coefficients of their spectral representations.

A third approach uses boosting [Jackson, 1997]. In this approach, basis functions are added iteratively. Each iteration, a basis function is selected that is highly correlated with a weighted version of the data. Initially, all examples have equal weight, so the first basis function added is correlated with the original data. Thereafter, examples that are classified incorrectly by the previously added basis functions receive more weight, so that basis functions added in subsequent iterations are more correlated with misclassified examples.

The search algorithms described in the following sections are presented with a boosting approach in mind, and each algorithm returns the largest coefficient found during search. However, with simple modification, they can be used to find any desired number of large coefficients, allowing them to be used with the all-large-coefficients Fourier learning approach.

2.3 Generalized Fourier-based Learning
A Fourier-based learning algorithm will likely be most effective when f has, or can be approximated well by, a sparse Fourier representation. Of course, not all functions can be approximated efficiently by a sparse Fourier representation. However, the Fourier learning approach can be generalized to allow other representations, such as wavelet representations [Donoho and Johnstone, 1994; 1995] or representations

based on ANDs and ORs of input features [Drake and Ventura, 2005]. The search algorithms presented in this paper can be used with other representations by modifying a procedure described in the following section.

3 Finding Spectral Coefficients
For a spectral learning algorithm that selects basis functions with large coefficients, the heart of the learning algorithm is the search algorithm used to find large coefficients. In a boosting approach, the key to success is being able to find one large coefficient (per iteration).

3.1 Finding Spectral Coefficients is Hard
As mentioned previously, the number of basis functions is exponential in the number of inputs to the function being learned. A simple brute force calculation of spectral coefficients to find the largest would require $O(|X| \cdot 2^n)$ time. ($O(|X|)$ time is required to compute a single coefficient, and there are 2^n coefficients to consider.) Meanwhile, the Fast Walsh Transform algorithm (a Boolean-input analogue to the Fast Fourier Transform) requires $O(n \cdot 2^n)$ time. Both of these approaches are practical only for small learning problems.

Most previous work in spectral learning has either been in small domains (where a brute-force approach is feasible), made assumptions about which coefficients might be large (which limits applicability), or relied on conditions that are not practical in many real-world situations (such as requiring an oracle that can be queried for examples during training). This paper considers algorithms for the more general learning scenario in which brute force computation may be infeasible, no assumptions about the coefficients can be made, and the set of training examples may be fixed prior to learning.

Unfortunately, it can be shown that the problem of finding a large spectral coefficient with respect to a set of examples is as hard as solving the MAX-2-SAT problem, and is therefore an NP-complete problem. The following theorem expresses this idea for the Fourier spectrum.

Theorem 1. *Let X be a set of $\langle x, f(x) \rangle$ pairs, where $x \in \{0, 1\}^n$ and $f(x) \in \mathbb{R}$. Given $\theta \in \mathbb{R}$, the problem of determining whether $\exists \alpha \in \{0, 1\}^n$ such that $|\hat{f}_X(\alpha)| \geq \theta$ is NP-complete.*

Proof Sketch. The proof is by reduction from MAX-2-SAT. The key observation in the proof is that every CNF expression can be converted in polynomial time to a set X of $\langle x, f(x) \rangle$ pairs such that each coefficient of $\hat{f}_X$ gives the number of clauses that will be satisfied by one of the possible truth assignments. Thus, solving the large Fourier coefficient problem for X would give the MAX-2-SAT solution for the CNF expression. $\square$

This coefficient search result generalizes to a class of spectral representations with certain properties.

3.2 Searching by Bounding Coefficient Size
Fortunately, although the NP-completeness result suggests that an efficient algorithm for finding large coefficients in arbitrary spectra does not exist, a technique for bounding coefficient size in any given region of the spectrum makes it

	CreateChild($parentNode, i, value$)
(1)	$child.\beta \leftarrow parentNode.\beta$
(2)	$child.\beta_i \leftarrow value$
(3)	$child.X_\beta \leftarrow \emptyset$
(4)	for each $\langle x, f(x) \rangle \in parentNode.X_\beta$
(5)	$v \leftarrow x$
(6)	$f(v) \leftarrow f(x)$
(7)	if $v_i = 1$
(8)	$v_i \leftarrow 0$
(9)	if $value = 1$
(10)	$f(v) \leftarrow -f(v)$
(11)	if $\exists \langle z, f(z) \rangle \in child.X_\beta$ such that $v = z$
(12)	$\langle z, f(z) \rangle \leftarrow \langle z, f(z) + f(v) \rangle$
(13)	else
(14)	$child.X_\beta \leftarrow child.X_\beta \cup \langle v, f(v) \rangle$
(15)	return $child$

Figure 1: Procedure for obtaining children of a Fourier coefficient search node. i is the digit of the parent node's label that is to be set, and $value$ is the value it is to be set to.

possible to find large coefficients much more efficiently than with the previously mentioned approaches. A Boolean function version of this technique was used in the context of a best-first search algorithm [Drake and Ventura, 2005]. Here we generalize the technique to handle real-valued functions and show that it can be incorporated into search algorithms that are capable of handling larger problems. In this section we will consider only the case of searching the Fourier (XOR) spectrum. With some modification this technique can be used to find coefficients in other spectra.

To explain the technique we will introduce some additional notation. Let $\beta \in \{0, 1, *\}^n$, where $*$ is a wildcard value, be a partially or fully defined basis function label that represents a region of the Fourier spectrum. Specifically, β represents the region of the spectrum consisting of all α such that $\forall i$ $(\beta_i = * \vee \alpha_i = \beta_i)$. We will use the notation $\alpha \in \beta$ to denote an α in region β.

Suppose that the coefficient search space is a tree of nodes with one node for each possible β. The search begins at the root node, which has label $\beta = *^n$. As the search proceeds, nodes are expanded by replacing them with child nodes whose labels have one of the wildcards set to either 0 or 1. Both children set the same wildcard, with one child taking the value 0 and the other taking the value 1. A leaf node, which will have no wildcards in its label, is a solution, and its label corresponds to the label of a specific basis function.

In addition to the label β, there is a set of examples, X_β, associated with each node that facilitates the computation of coefficient bounds by implicitly storing in child nodes information obtained while computing bounds for parent nodes. For the root node, $X_\beta = X$. For any other node, X_β is derived from its parent by the procedure shown in Figure 1. Using this procedure, the size of the largest possible Fourier coefficient in region β can be bounded as follows:

$$\max_{\alpha \in \beta} \left| \hat{f}_X(\alpha) \right| \leq \frac{\sum_{\langle x, f(x) \rangle \in X_\beta} |f(x)|}{|X|} \quad (6)$$

To give some intuition behind this technique, note that the

largest possible Fourier coefficient for a given X is given by

$$\max_{\alpha \in *^n} \left| \hat{f}_X(\alpha) \right| \leq \frac{\sum_{\langle x, f(x) \rangle \in X} |f(x)|}{|X|}$$

which is the coefficient bound of the root node. This maximum correlation exists only if there is an α such that either $sign(f(x)) = sign(\chi_\alpha(x))$ or $sign(f(x)) \neq sign(\chi_\alpha(x))$ for all $\langle x, f(x) \rangle \in X$. To the extent that a basis function, or set of basis functions, does not exhibit either of these correlations, coefficient size drops.

The procedure described in Figure 1 captures this idea at line 12, where examples are merged. Examples are merged here only if they are identical on inputs for which β_i has not yet been set. This can be done because the effect of inputs for which β_i has been set has already been taken into account (by inverting outputs, at line 10, whenever $\beta_i = v_i = 1$). When examples are merged, the coefficient bound may decrease since $|f(x) + f(v)| < |f(x)| + |f(v)|$ when $sign(f(x)) \neq sign(f(v))$.

Note that for a solution node, where $\beta \in \{0, 1\}^n$, all examples will have merged into one, and the coefficient of χ_β is given by

$$\hat{f}_X(\beta) = \frac{\sum_{\langle x, f(x) \rangle \in X_\beta} f(x)}{|X|}$$

while the magnitude of the coefficient is given by the absolute value of that quantity.

The following sections describe search algorithms that all use this coefficient bounding technique but explore the search space in different ways. Two of the methods are complete, while the other is an incomplete search technique.

4 Complete Search Techniques

The complete search algorithms presented here always find the largest coefficients, but may require exponential time and/or memory to do so. The first is a previously introduced best-first search algorithm [Drake and Ventura, 2005], and the second is a branch-and-bound search algorithm. Empirical results show that the branch-and-bound algorithm can find the largest coefficient in about the same amount of time as the best-first algorithm, while requiring far less memory. With a boosting approach in mind, both algorithms are presented here as if the task were to find the single largest coefficient. With simple modification, the algorithms can be used to find the r largest coefficients or to find all coefficients larger than a threshold, which would be done in the find-all-large-coefficients approach.

4.1 Best-First Search

The best-first search algorithm is outlined in Figure 2. Like the other search algorithms, it begins at the root node. After expanding the first node, however, it explores the space in order of $\max_{\alpha \in \beta} |\hat{f}_X(\alpha)|$, where β is the node's label. Nodes are stored in a priority queue in which the highest priority element is the node with the largest coefficient bound.

Since nodes are visited in best-first order, relatively few nodes are visited unnecessarily. However, the entire frontier of the search must be stored in memory, which can exhaust resources fairly quickly if the search becomes large.

```
FindLargeCoef-BestFirst(X)
    initialNode.β ← *ⁿ
    initialNode.X_β ← X
    priorityQueue.insert(initialNode)
    while priorityQueue.front() is not a solution
        node ← priorityQueue.removeFront()
        i ← GetInputToSetNext(node)
        priorityQueue.insert(CreateChild(node, i, 0))
        priorityQueue.insert(CreateChild(node, i, 1))
    return priorityQueue.front().β
```

Figure 2: The best-first search algorithm. Nodes are stored in a priority queue that always places the node with the largest coefficient bound at the front.

```
FindLargeCoef-BranchAndBound(X)
    initialNode.β ← *ⁿ
    initialNode.X_β ← X
    stack.push(initialNode)
    while stack is not empty
        node ← stack.pop()
        if max_{α∈node.β} |f̂_X(α)| > |f̂_X(α_best)|
            if node is a solution
                α_best ← node.β
            else
                i ← GetInputToSetNext(node)
                stack.push(CreateChild(node, i, 1))
                stack.push(CreateChild(node, i, 0))
    return α_best
```

Figure 3: The branch-and-bound search algorithm. Nodes are visited depth-first, and nodes whose coefficient bound is below the largest coefficient found so far ($|\hat{f}_X(\alpha_{best})|$) are pruned from the search.

4.2 Branch-and-Bound Search

The branch-and-bound search algorithm is outlined in Figure 3. It is a depth-first search in which search paths are pruned whenever a node's coefficient bound ($\max_{\alpha\in\beta}|\hat{f}_X(\alpha)|$) is below the best solution found so far ($|\hat{f}_X(\alpha_{best})|$). Figure 3 illustrates the use of a stack to perform this search, although it can be implemented using recursion as well.

The branch-and-bound algorithm will tend to visit more nodes than the best-first algorithm, but it has less overhead, so it can visit more nodes per second. In addition, its memory usage is linear in n, the number of inputs. If examples at each node are stored in a hash table of size h, then the algorithm's space complexity is $O(n(|X| + h))$. By comparison, the space complexity of the best-first algorithm is $O(m(|X| + h))$, where m, $n \leq m \leq 2^n$, is the number of nodes expanded during search. (For every node expanded and removed from the queue, two are added, so the queue size increases by one each time a node is expanded.)

4.3 Variable Ordering

In both algorithms, when a node is expanded, an input i for which $\beta_i = *$ is selected to be set to 0 and 1 in the child nodes (as indicated by the GetInputToSetNext function in Figures 2 and 3). The choice of i does not affect the completeness of

Table 1: Data set summary.

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

either algorithm, so inputs could be processed in an arbitrary order. However, both algorithms benefit greatly from a dynamic variable ordering scheme.

The variable ordering heuristic used in the experiments that follow selects an input according to the following:

$$\underset{i}{\operatorname{argmin}} \left(\max_{\alpha\in\beta_{i\leftarrow0}} \left|\hat{f}_X(\alpha)\right| + \max_{\alpha\in\beta_{i\leftarrow1}} \left|\hat{f}_X(\alpha)\right| \right) \quad (7)$$

in which $\beta_{i\leftarrow0}$ and $\beta_{i\leftarrow1}$ denote the labels of the child nodes that result from setting β_i to 0 and 1, respectively. This heuristic chooses the input that results in the tightest (i.e., smallest) combined coefficient bounds in the children. By obtaining tighter bounds more quickly, it is possible to more quickly determine which portions of the space can be ignored.

4.4 Complete Search Results

This section compares the performance of the algorithms on several data sets [Newman *et al.*, 1998], which are summarized in Table 1. Each data set represents a Boolean classification problem. In each data set, non-Boolean input features were converted into Boolean features. Each numeric input feature was converted into a single Boolean feature that indicated whether the value was above or below a threshold. 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 would be true in an example. The number of inputs listed in Table 1 is the number of inputs after converting non-Boolean inputs to Boolean.

Tables 2-4 compare the performance of the best-first and branch-and-bound algorithms, both with and without the variable ordering heuristic, when used to find the largest coefficient for each of the data sets. Tables 2, 3, and 4 show the average number of nodes expanded, the average amount of time required, and the average memory usage, respectively. (Note that for two of the data sets the best-first algorithm without the variable ordering heuristic ran out of memory, so results are not displayed in those two cases.)

As the node counts in Table 2 show, the variable ordering heuristic drastically reduces the number of nodes visited by both algorithms on the larger search problems. Table 3 shows that this improvement in the number of visited nodes leads to smaller run times as well, in spite of the added computation required to compute the heuristic.

Comparing the best-first and branch-and-bound algorithms, Table 2 shows that the best-first algorithm typically

Table 2: Average number of nodes visited while searching for the largest Fourier coefficient. The variable ordering heuristic (H) reduces the number of nodes visited by both algorithms, while the best-first (BFS) algorithm usually visits fewer nodes than the branch-and-bound (B&B) algorithm.

Data Set	B&B	BFS	B&B+H	BFS+H
Chess	432,986.1	-	2,012.5	296.0
German	4,097.8	3,912.2	155.0	135.5
Heart	4,072.8	3,935.4	196.4	184.4
Pima	33.4	20.0	33.8	13.3
SPECT	8,007.9	8,040.8	1,170.2	1,170.7
Voting	52.2	28.3	26.0	26.0
WBC1	24.8	21.6	17.0	17.0
WBC2	252,833.5	-	479.3	339.3
WBC3	6,936.2	6,500.7	106.1	101.8

Table 3: Average time (in seconds) to find the largest Fourier coefficient. Although the best-first algorithm (BFS) usually visits fewer nodes, its run time is roughly equivalent to that of the branch-and-bound (B&B) algorithm.

Data Set	B&B	BFS	B&B+H	BFS+H
Chess	242.16	-	1.39	0.91
German	1.11	1.94	0.11	0.12
Heart	0.40	0.58	0.04	0.05
Pima	0.00	0.00	0.00	0.00
SPECT	1.11	2.08	0.42	0.46
Voting	0.00	0.00	0.00	0.00
WBC1	0.00	0.00	0.00	0.00
WBC2	23.36	-	0.11	0.12
WBC3	1.48	2.55	0.05	0.05

visits fewer nodes. However, Table 3 shows that the algorithms require about the same amount of time to find a solution, so the additional overhead of the best-first algorithm appears to offset the potential gain of visiting fewer nodes. Table 4 demonstrates the branch-and-bound algorithm's memory advantage. Perhaps more important than the differences observed here, however, is the fact that the branch-and-bound algorithm's worst-case memory complexity is linear in n, while the best-first search algorithm's is exponential.

5 Incomplete Search Techniques

An alternative to the previously introduced complete search algorithms is an incomplete algorithm that may not always find the largest coefficients but is guaranteed to finish its search quickly and usually finds good solutions.

5.1 Beam Search

The beamed breadth-first search described in Figure 4 explores the search space 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.

The number of nodes that will be considered by the beam search algorithm is $O(nk)$, where k is the width of the beam. Thus, unlike the complete algorithms, its worst-case time complexity is not exponential in n. Its space complexity is

Table 4: Average memory usage (in terms of the number of nodes) during a search for the largest Fourier coefficient. The branch-and-bound (B&B) algorithm's memory usage is proportional to the number of inputs, and tends to be much less than that of the best-first algorithm (BFS), whose memory usage is proportional to the number of visited nodes.

Data Set	B&B/B&B+H	BFS	BFS+H
Chess	39	-	297.0
German	25	3,913.2	136.5
Heart	21	3,936.4	185.4
Pima	9	21.0	14.3
SPECT	23	8,041.8	1,170.7
Voting	17	29.3	27.0
WBC1	10	22.6	18.0
WBC2	33	-	340.3
WBC3	31	6,501.7	102.8

```
FindLargeCoef-BeamSearch(X, k)
    initialNode.β ← *ⁿ
    initialNode.X_β ← X
    bestNodes ← ∅ ∪ initialNode
    for j ← 1 to n
        priorityQueue ← ∅
        for each node ∈ bestNodes
            i ← GetInputToSetNext(node)
            priorityQueue.insert(CreateChild(node, i, 0))
            priorityQueue.insert(CreateChild(node, i, 1))
        bestNodes ← ∅
        while bestNodes.size() ≤ k
            node ← priorityQueue.removeFront()
            bestNodes ← bestNodes ∪ node
    return bestNodes.best().β
```

Figure 4: The beam search algorithm. The search space is explored breadth-first, with all but the k most promising nodes at each level being pruned.

$O(k(|X| + h))$, where, as before, h is the size of the hash table containing the examples in a node.

Like the best-first and branch-and-bound algorithms, the beam search algorithm can use an arbitrary variable ordering, but it can benefit from a good heuristic. Here, the motivation for the heuristic is different. In the case of the compete algorithms, the heuristic is used to reduce the number of nodes in the search space that need to be visited and has no effect on the solution. For the beam search, however, the number of nodes that will be considered is fixed, and the heuristic is used to improve the solution.

Without a heuristic, it is possible for early decisions to be made on inputs for which there is little information, meaning that partially-defined labels that lead to good solutions might get pruned before the inputs that would reveal their usefulness were considered. The variable selection heuristic defined previously (Equation 7) favors inputs that tighten the coefficient bounds in the child nodes, making it less likely for good nodes to be pruned.

Table 5: 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.

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

5.2 Incomplete Search Results

Table 5 shows the result of attempting to learn functions with a single basis function that is found by a beam search. 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 be very wide before the beam search does about as well as a complete search.

In fact, sometimes the beam search performs better. 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 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.

Table 6 shows that the beam search can find its solutions in less time than the branch-and-bound algorithm. The computational advantage of the beam search will be more important when spectral techniques are applied to higher dimensional problems involving hundreds or thousands of input features. Preliminary experiments with natural language processing tasks involving thousands of input features suggest that the beam search approach can still be used to find good solutions long after it has become infeasible to compute exact solutions.

6 Conclusion

In this paper we have considered the problem of finding large coefficients in the Fourier spectrum, which is a central task in Fourier-based learning. We have described a technique for efficiently bounding Fourier spectra that can be incorporated into different types of search algorithms.

Empirical results show that a complete branch-and-bound algorithm based on the technique outperforms a previously introduced best-first algorithm by finding solutions in the

Table 6: Average training time (in seconds) of the spectral learning algorithm when using the branch-and-bound and beam search algorithms to find coefficients. On the larger problems (shown here), the beam search is much faster, and its solutions (see Table 5) are usually equally good. (On the smaller problems the difference in time is negligible.)

Data Set	B&B+H	Beam (width = 8)
Chess	1.49	0.23
German	0.12	0.04
Heart	0.04	0.01
SPECT	0.48	0.02
WBC2	0.13	0.02

same amount of time while using less memory. Meanwhile, experiments with an incomplete beam search algorithm demonstrate that even using a small beam width it is possible to find solutions that result in learning accuracy comparable to that obtained by a complete algorithm.

References

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

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

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

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

[Kargupta et al., 1999] 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.

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

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

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

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