Advances in Adaptive Data Analysis
Vol. 2, No. 1 (2010) 97–114

World Scientific
www.worldscientific.com

A SPARSE GREEDY SELF-ADAPTIVE ALGORITHM FOR CLASSIFICATION OF DATA

ANKUR SRIVASTAVA

Mechanical Engineering and Material Science Department
William Marsh Rice University, 6100 Main Street
Houston, Texas 77005-1827, Mail Stop 124
ankur.srivastava@rice.edu

ANDREW J. MEADE

Mechanical Engineering and Material Science Department
William Marsh Rice University, 6100 Main Street
Houston, Texas 77005-1827, Mail Stop 321
meade@rice.edu

Kernels have become an integral part of most data classification algorithms. However, the kernel parameters are generally not optimized during learning. In this work a novel adaptive technique called Sequential Function Approximation (SFA) has been developed for classification that determines the values of the control and kernel hyper-parameters during learning. This tool constructs sparse radial basis function networks in a greedy fashion. Experiments were carried out on synthetic and real-world data sets where SFA had comparable performance to other popular classification schemes with parameters optimized by an exhaustive grid search.

Keywords: Greedy algorithms; radial basis function; method of weighted residuals; control parameters; statistical significance; sparsity.

1. Introduction

Pattern recognition is widely studied in the area of machine learning research. This subject studies tools capable of conducting data analysis and knowledge extraction on real-world noisy data. Popular pattern recognition algorithms include back-propagation networks [Cun *et al.* (1990)], k-nearest neighbor [Dasarathy (1991)], classification trees [Breiman *et al.* (1984)], support vector machines [Scholkopf and Smola (2002)], and radial basis function networks [Buhmann (2003)]. However, a major drawback of the training and testing approach used by the aforementioned data classification tools results from the dependence on user-determined control parameters. Most classifiers require the use of control parameters and/or kernel hyper-parameters. A widely accepted practice to determine the values of these parameters is to perform a grid search over the training or validation set which is necessary to find optimal values of the control and/or kernel hyper-parameters.

Consequently, the optimum performance of the classifier depends heavily on the user-controlled parameters and is difficult to obtain. To address this performance problem, a self-adaptive classifier has been developed that determines the values of the control and kernel parameters during learning. Such a classification tool would assign different parameters to each basis function, thereby reducing the dependence of the data classification accuracy on any initial choice of control and kernel hyper-parameters.

In this work we focus on constructing Gaussian radial basis function networks, however, this technique can be applied to a number of basis functions like hyperbolic tangents, polynomials, B-splines, and trigonometric functions. Owing to its greedy nature the algorithm constructs sparse radial basis function networks. Sparsity provides redundancy and robustness to the nonlinear model and is directly related to the generalization capacity of the model [Grapel *et al.* (2000)].

In this work we propose a self-adaptive data classification technique that estimates the width and center of each Gaussian radial basis function during training. It does not need a validation set to estimate the learning parameters. To evaluate the performance of our algorithm we compare the classification accuracies against popular approaches like support vector machines and nearest neighbor classifiers on simulated and real-world data sets with their parameters optimized by an exhaustive grid search.

The developed classifier is called Sequential Function Approximation (SFA) and is based on Method of Weighted Residual interpretation of a greedy sparse approximation technique. The remainder of this paper is organized as follows. In Sec. 2 a review of other related sequential approximation methods is presented. In Sec. 3 the algorithm of SFA is presented with its implementation issues. In Sec. 4 results are shown and experimental comparisons of SFA with other standard classifiers. Conclusions and future avenues of research are discussed in Sec. 5.

2. Related Algorithms

In this section we present the links between SFA and other sequential approximation algorithms present in the literature. Greedy sequential methods have been an area of active research for many years in the fields of signal processing, neural networks, kernel methods, and statistics. Sequential greedy approximation methods construct a series of locally optimum solutions in the spirit of obtaining the global optimum.

2.1. *Greedy algorithms*

Greedy algorithms are constructive approximation algorithms that decompose the target vector into a linear sum of basis functions. These algorithms are highly nonlinear because they use the target vector to determine the coefficients of the linear sum and to choose the next basis function. We focus on function approximation algorithms that construct models of the form $y_n^a(x) = \sum_{i=1}^{n} c_i(\phi(x, \beta_i) + b_i)$. For classification problems a proxy function, for example

$y_n^a(x) = \text{sign}(\sum_{i=1}^{n} c_i(\phi(x, \beta_i) + b_i))$, can be used. Here $x \in \mathbb{R}^d$ denotes the input vector, $y \in \{-1, 1\}$ denotes the target, and $c_i, b_i \in \mathbb{R}$ are the linear coefficient and the bias, respectively. y_n^a is the approximation of the target vector after the addition of n basis functions. The basis functions $\phi(x, \beta_i)$ are Gaussian radial basis functions with kernel parameters β_i that include the basis center x_i^* and the width σ_i. For the remainder of this work we will assume the width to be uniform in all d dimensions. To construct the best n-term approximation of the target vector, we need to solve for four unknowns, namely, c_i, b_i, σ_i, and x_i^* for each basis function. This would yield a $4n$ dimensional optimization problem assuming that the basis centers are chosen from the available training points. A greedy function approximation approach allows us to solve for all of the four unknowns for each basis function at a time.

Typical sparse kernel modeling techniques simplify this problem by fixing the width of the basis functions, thereby reducing the number of optimization problems to be solved. A greedy function approximation algorithm picks a basis function from a given set of basis functions, calculates the residual error of the approximation, and uses the residual to pick the next basis function. Counterparts of greedy algorithms exist as Matching Pursuit [Mallat and Zhang (1993)] in the signal processing community and Boosting [Freund and Schapire (1996)] in the statistics community. The given collection of basis functions is often called a dictionary. The Matching Pursuit algorithm chooses a basis function from the dictionary that maximizes the absolute inner product of the basis function $\vec{\phi}_n$ with the residual vector $\vec{r}_{n-1}$. Geometrically this can be interpreted as choosing a $\vec{\phi}_n$ that makes the most acute angle with $\vec{r}_{n-1}$ in $\mathbb{R}^s$, where s is the number of training points. Here we represent the nth basis function vector $\vec{\phi}_n = \{\phi(x_1, \beta_n), \phi(x_2, \beta_n), \ldots, \phi(x_s, \beta_n)\}$ compactly as $\vec{\phi}_n$ and the residual vector at nth stage $\vec{r}_n = \{r_n(x_1), r_n(x_2), \ldots, r_n(x_s)\}$ compactly as $\vec{r}_n$. In Sec. 2.2 we stress on the differences between SFA and Matching Pursuit algorithms in order to throw light on some of the key strengths of SFA. Later in Sec. 4 we compare SFA against these algorithms in order to corroborate our points with experimental results. In Secs. 2.3 and 2.4 we present some other greedy approximation techniques that share some similarities with SFA. Comparison of SFA with these algorithms will be a part of the future work.

2.2. *Kernel matching pursuit*

Vincent and Bengio (2002) showed how Matching Pursuit could be used to build a kernel-based solution to a machine learning problem and called it Kernel matching pursuit (KMP). It is a greedy algorithm for building an approximation of a discriminant function as a linear combination of basis functions chosen from a kernel-induced dictionary. Vincent and Bengio (2002) compared the performance of KMP with SVM and radial basis function (RBF) networks on several real-world data sets. They concluded that KMP gave comparable results to SVMs with sparser

approximations. However, Suykens *et al.* (2002) showed that sparser solutions can be obtained by SVMs *via* a pruning strategy.

The proposed algorithm bears some similarities with the basic version of KMP but the primary differences are the following. In KMP the basis function at the nth stage is chosen such that it maximizes $abs(\frac{\langle \vec{r}_{n-1}, \vec{\phi}_n \rangle}{\langle \vec{\phi}_n, \vec{\phi}_n \rangle})$. Since a constant width is chosen and the basis functions are centered on the training points, choosing a new basis function amounts to choosing a basis center from the available training data based on the above-mentioned criterion. From a geometrical perspective such an approach will choose the basis function $\vec{\phi}_n$ that is most aligned with the residual vector $\vec{r}_{n-1}$ in $\mathbb{R}^s$. An improvement is possible if the width of the basis functions is not fixed. Allowing the width to vary would increase the size of the dictionary generated by the training points. A larger dictionary allows us to choose a better basis function that maximizes the above-mentioned criterion.

In the proposed approach we optimize for the spread of each basis function and choose the basis center corresponding to the maximum absolute value of the residual at that stage. This can be interpreted as choosing basis functions from a larger dictionary that are aligned more parallel to $\vec{r}_{n-1}$ in $\mathbb{R}^s$. Choosing a new basis function in this manner will diminish more of the residual error and results in sparser approximations as compared to KMP.

The basic version of KMP did not perform to satisfaction in the experiments done by Vincent and Bengio (2002). To address those limitations the authors proposed a back-fitting and a pre-fitting version of KMP. Properties of orthogonal projections were used in the back-fitting and pre-fitting versions of KMP to update all previously chosen basis function and coefficients of linear sum. The aim of the back-fitting and pre-fitting versions of KMP was to select a basis function that is the most aligned to the residual vector and is the most perpendicular to the previously chosen basis function. Keeping the basis centers fixed at the training points and the width of the RBF fixed does not always result in a good choice of basis function from the available dictionary. In this work we show that allowing the width of the basis function to vary will result in a better choice of the basis function because we get one more degree of freedom to choose the basis function. In Sec. 3 we compare SFA against the basic version of KMP and the pre-fitting version of KMP on artificial and real-world classification problems. The results show that SFA obtained sparser classifiers with similar classification accuracies on almost all data sets. Moreover, only one pass through the dictionary was required by SFA to choose the basis function as opposed to the pre-fitting version of KMP that required two passes.

2.3. *Sparse approximation methods*

Sparse approximation methods broadly fall into the following broad categories: (a) sequential forward greedy algorithms, (b) backward greedy algorithms, and (c) mathematical programming approaches [Nair *et al.* (2002)].

KMP and SFA are examples of sequential forward greedy algorithms. Other popular examples are Order Recursive Matching Pursuit (ORMP) [Natarajan (1995)] and Orthogonal Least Squares RBF (OLS-RBF) [Chen *et al.* (1991)]. Vincent and Bengio (2002) described the pre-fitting version of KMP with squared error loss and Gaussian kernel to be identical to the OLS-RBF algorithm. They compared OLS-RBF against Gaussian Support Vector Machines (SVMs) to conclude that OLS-RBF performed as well as Gaussian SVM with sparser approximations. One of the contributions of this work is to compare the performance of Least Square Support vector Machines (LS-SVM) against KMP algorithms and SFA. The primary difference between ORMP and SFA is that ORMP normalizes and reorients all unselected basis functions and avoids the recycling problem by choosing from previously unselected basis functions. SFA avoids the recycling problem by assigning a different width to each basis function. So even if the center selection routine makes a poor choice of the basis center from the available training points, because of the width optimization step a different and a relatively better basis function is selected at each step and the recycling problem is avoided. Some other greedy algorithms in the spirit of Natarajan's algorithm have also been published [Nair *et al.* (2002)]. But in these works the question of dependence of classification accuracy on the width of the Gaussian radial basis functions has not been addressed.

The backward greedy algorithms [Couvreur and Bresler (1999)] start with a $s \times s$ Gram matrix, where s is the number of observations in the training set and iteratively eliminates columns. Even though this helps in achieving guaranteed convergence properties it is more computationally expensive than the forward algorithms. The mathematical programming approaches like Basis Pursuit differs from the sequential forward greedy algorithms in the sense that they use regularization in the cost function to determine a sparse solution. Basis pursuit also employs quadratic programming like support vector machines to minimize the associated cost function. Girosi (1998) proposed a modified version of Basis Pursuit De-noising [Chen (1995)] and showed that SVMs are equivalent to them as they solve the same quadratic programming problem. However, sequential forward greedy algorithms like SFA and KMP enforce regularization *via* sparsity.

2.4. *Other related methods*

In the statistics community the ideas of greedy learning are popular by the name of Boosting [Freund and Schapire (1996)] and Projection Pursuit [Friedman and Stuetzle (1984)]. KMP in its basic form could be seen very similar to Boosting if the weak learners were seen as the kernel functions centered on the training points. The Projection Pursuit method constructs the approximation of the target vector as a linear sum of ridge functions in a forward stagewise manner. The input training points are projected on a unit direction vector which is optimized at each stage. A ridge function which results in the maximum reduction of the residual error is selected. The addition of ridge functions continues till the residual error is

smaller than a user-defined threshold. This method bears clear similarities with the Matching Pursuit methods of the signal processing community.

In the neural network community constructing the neural architecture in a sequential forward manner goes by the name of Cascade-correlation [Fahlman and Labiere (1990)]. A new hidden unit is added if the previously added neurons do not reduce the residual error of the network below a specified threshold. The algorithm tries to maximize the magnitude of correlation between the new unit's output and the residual error signal. With these algorithms the users do not need to worry about the size and topology of the network. Other constructive algorithms include Dynamic node creation [Wang *et al.* (1994)] and the Resource-allocating network [Yingwei *et al.* (1997)].

3. Sequential Function Approximation

We start our approximation of the unknown target function y by noting that a continuous d-dimensional function can be arbitrarily well-approximated by a linear combination of radial basis functions ϕ. Sequential Function Approximation (SFA) was developed from mesh-free finite element research but shares similarities with the Boosting and Matching Pursuit algorithms. We start our approximation of the target vector y utilizing the Gaussian radial basis function ϕ.

$$\begin{aligned} y_n^a &= \sum_{i=1}^{n} c_i(\phi(x,\beta_i)+b_i) \quad \text{with} \\ \phi(x,\beta_i) &= \exp[\log[\alpha_i](x-x_i^*)\bullet(x-x_i^*)] \quad \text{for } 0<\alpha_i<1. \end{aligned} \tag{1}$$

Traditionally, Gaussian radial basis functions are written as $\phi(x,\beta_i) = \exp[-\frac{(x-x_i^*)\cdot(x-x_i^*)}{\sigma_i^2}]$. We write the radial basis function as Eq. (1) in order to set up the optimization problem for α_i as a bounded nonlinear line search instead of an unconstrained minimization problem for σ_i. The basic principles of our greedy algorithm are motivated by the similarities between the iterative optimization procedures by Jones (1990; 1992) and Barron (1993) and the Method of Weighted Residuals (MWR), specifically the Galerkin method [Fletcher (1984)]. We can write the function residual vector $\vec{r}_n$ at the nth stage of approximation as in Eq. (2):

$$\begin{aligned} \vec{r}_n &= y(x) - y_n^a(x) = y(x) - y_{n-1}^a(x) - c_n(\phi(x,\beta_n)+b_n) \\ &= \vec{r}_{n-1} - c_n(\phi(x,\beta_n)+b_n) = \vec{r}_{n-1} - c_n(\vec{\phi}_n + b_n). \end{aligned} \tag{2}$$

Using the Petrov–Galerkin approach, we select a coefficient c_n that will force the function residual to be orthogonal to the basis function and b_n using the discrete inner product $\langle , \rangle_D$ given by Eq. (3):

$$\begin{aligned} \vec{r}_n \bullet (\vec{\phi}_n + b_n) &= \sum_{j=1}^{s} r_n(x_j)(\phi_n(x_i,\beta_n)+b_n) = \langle \vec{r}_n, (\vec{\phi}_n + b_n)\rangle_D = -\left\langle \vec{r}_n, \frac{\partial \vec{r}_n}{\partial c_n} \right\rangle_D \\ &= -\frac{1}{2}\frac{\partial \langle \vec{r}_n, \vec{r}_n\rangle_D}{\partial c_n} = 0, \end{aligned} \tag{3}$$

which is equivalent to selecting a value of c_n that will minimize $\langle \vec{r}_n, \vec{r}_n \rangle_D$ or

$$c_n = \frac{(s\langle \vec{\phi}_n, \vec{r}_{n-1} \rangle_D - \langle \vec{r}_{n-1} \rangle_D \langle \vec{\phi}_n \rangle_D)}{(s\langle \vec{\phi}_n, \vec{\phi}_n \rangle_D - \langle \vec{\phi}_n \rangle_D \langle \vec{\phi}_n \rangle_D)} \tag{4}$$

with

$$b_n = \frac{(\langle \vec{r}_{n-1} \rangle_D \langle \vec{\phi}_n, \vec{\phi}_n \rangle_D - \langle \vec{\phi}_n \rangle_D \langle \vec{\phi}_n, \vec{r}_{n-1} \rangle_D)}{(s\langle \vec{\phi}_n, \vec{r}_{n-1} \rangle_D - \langle \vec{r}_{n-1} \rangle_D \langle \vec{\phi}_n \rangle_D)}. \tag{5}$$

The discrete inner product $\langle \vec{r}_n, \vec{r}_n \rangle_D$, which is equivalent to the square of the discrete L_2 norm, can be re-written, with the substitution of Eqs. (4) and (5), as

$$\sum_{j=1}^{s} r_n(x_j) r_n(x_j) = \|\vec{r}_n\|_{2,D}^2 = \langle \vec{r}_n, \vec{r}_n \rangle_D$$

$$\langle \vec{r}_n, \vec{r}_n \rangle_D = \left\langle \vec{r}_{n-1} - \frac{\langle \vec{r}_{n-1} \rangle_D}{s}, r_{n-1} - \frac{\langle \vec{r}_{n-1} \rangle_D}{s} \right\rangle_D \times \left(1 - \frac{\langle \vec{r}_{n-1} - \frac{\langle \vec{r}_{n-1} \rangle_D}{s}, \vec{\phi}_n - \frac{\langle \vec{\phi}_n \rangle_D}{s} \rangle_D^2}{\langle \vec{\phi}_n - \frac{\langle \vec{\phi}_n \rangle_D}{s}, \vec{\phi}_n - \frac{\langle \vec{\phi}_n \rangle_D}{s} \rangle_D \langle \vec{r}_{n-1} - \frac{\langle \vec{r}_{n-1} \rangle_D}{s}, \vec{r}_{n-1} - \frac{\langle \vec{r}_{n-1} \rangle_D}{s} \rangle_D} \right). \tag{6}$$

Recalling the definition of the cosine given by Eq. (7), using arbitrary functions f and v and the discrete inner product,

$$\cos(\theta) = \frac{\langle f, v \rangle_D}{\langle f, f \rangle_D^{1/2} \langle v, v \rangle_D^{1/2}}. \tag{7}$$

Equation (6) can be written as

$$\begin{aligned} \|\vec{r}_n\|_{2,D}^D &= \left\langle \vec{r}_{n-1} - \frac{\langle \vec{r}_{n-1} \rangle_D}{s}, \vec{r}_{n-1} - \frac{\langle \vec{r}_{n-1} \rangle_D}{s} \right\rangle \sin^2(\theta_n) \\ &= \left(\|\vec{r}_{n-1}\|_{2,D}^D - \frac{\langle \vec{r}_{n-1} \rangle_D^2}{s} \right) \sin^2(\theta_n), \end{aligned} \tag{8}$$

where θ_n is the angle between $\vec{\phi}_n$ and $\vec{r}_{n-1}$ since $\frac{\langle \vec{r}_{n-1} \rangle_D}{s}$ and $\frac{\langle \vec{\phi}_n \rangle_D}{s}$ are scalars. With Eq. (8) we note that $\|\vec{r}_n\|_{2,D}^2 < \|\vec{r}_{n-1}\|_{2,D}^2$ as long as $\theta_n \neq \pi/2$, which is a very robust condition for convergence. By inspection, the minimum of Eq. (8) is $\theta_n = 0$, implying Eq. (9):

$$c_n(\phi(x_i, \beta_n) + b_n) = r_{n-1}(x_i) \quad \text{for } i = 1, \ldots, s. \tag{9}$$

Therefore, to force $\|\vec{r}_n\|_{2,D}^2 \to 0$ with as few stages n as possible, a low-dimensional function approximation problem must be solved at each stage. This involves a bounded nonlinear minimization of Eq. (6) to determine the two variables α_i ($0 < \alpha_i < 1$) and index j^* ($x_n^* = x_j \in \mathbb{R}^d$) for the basis function center taken from the

training set. The dimensionality of the nonlinear optimization problem is kept low since we are solving only one basis at a time.

3.1. *Implementation of the algorithm*

Though our SFA scheme allows the basis center (x_n^*) to be located anywhere in $\mathbb{R}^d$, the practical application to problems with multiple inputs constrains the centers to the set of sample points $\{x_1, \ldots, x_s\}$. At each stage we determine x_n^* such that $|\vec{r}_{n-1}(x_n^*)| = \max|\vec{r}_{n-1}|$. The remaining optimization variable α_n is continuous and constrained to $0 < \alpha_n < 1$.

A benefit to using radial basis functions is that in practical applications we can ignore the denominator in the discrete inner product formulation of Eq. (6). As a result, the determination of β_n requires only

$$\min_{\beta_n} \left[- \left\langle \vec{r}_{n-1} - \frac{\langle \vec{r}_{n-1} \rangle_D}{s}, \vec{\phi}_n - \frac{\langle \vec{\phi}_n \rangle_D}{s} \right\rangle_D^2 \right]. \tag{10}$$

Optimization of Eq. (6) with respect to σ_n is a non-convex and an unconstrained nonlinear optimization problem. But formulating the basis function as shown in Eq. (1) allows us to formulate the optimization problem for α_i as a bounded line search problem. Traditional bracketing methods such as Golden selection search algorithms coupled with parabolic interpolation methods can be used for such problems. An advantage of such an approach is that it does not require any initial value of the optimization parameter but is based on the approximation that the objective function is unimodal in the prescribed interval $0 < \alpha_i < 1$. However, in this problem, assuming unimodality is safe because the interval is small. The algorithm terminates when $\max|\vec{r}_n| \leq \tau$, where τ is the tolerance desired by the user. For binary classification, where $y = +1$ or -1, we set $\tau = 1.0$. Setting an upper bound on the maximum absolute value of the residual balances the empirical risk and the complexity of the model thereby preventing overfitting. In the case of a classification problem setting a tolerance value of 1.0 suffices because unlike regression problems the classifier needs to learn only the sign of the approximation.

In this work multi-class classification problems are tackled by applying several binary classifiers in parallel. Approaches to combining binary classifiers include the *one vs. one* combination, the *one vs. all* pairing, and the error correcting output coding. There is no one best approach to combine several binary classifiers. Even though *one vs. all* has its own shortcomings, in this work this approach was used because of its simplicity and the widely accepted fact that it often produces good results. Rifkin and Klautau (2004) compared the *one vs. all* approach with other existing methods and concluded that *one vs. all* classification will produce results as good as any other approach if the underlying classifiers are well-tuned. Comparison of other methods with combine binary classifiers deserves a separate study.

To implement the SFA algorithm the user takes the following steps:

Sequential Function Approximation

1. Initiate the algorithm with the labels of the training data $\vec{r}_0 = \{y(x_1), \ldots, y(x_s)\}$.
2. **Basis center selection:** Search the components of $\vec{r}_{n-1}$ for the maximum magnitude. Record the component index j^*.
3. $x_n^* = x_{j^*}$.
4. **Basis width optimization:** With $\vec{\phi}_n$ centered at x_n^*, minimize Eq. (10) using optimization methods for bounded line search problems.
5. **Coefficient calculation:** Calculate the coefficients c_n and b_n from Eqs. (4) and (5), respectively.
6. **Residual update:** Update the residual vector $\vec{r}_n = \vec{r}_{n-1} - c_n(\vec{\phi}_n + b_n)$. Repeat Steps 3–6 until the termination criterion $\max|\vec{r}_n| \leq 1.0$ has been met.
7. Use the constructed binary classifier to predict on the test set.
8. Repeat Steps 1–7 for remaining classes.
9. Use *one vs. all* scheme to obtain the final prediction on the test set.

Our binary method is linear in storage with respect to s since it needs to store only $s + sd$ vectors to compute the residuals: one vector of length $s(\vec{r}_n)$ and s vectors of length $d(x_1, \ldots, x_s)$ where s is the number of samples and d is the number of dimensions. To generate the SFA model requires two vectors of length $n(\{c_1, \ldots, c_n\}$ and $\{b_1, \ldots, b_n\})$ and n vectors of length $n + 1(\beta_1, \ldots, \beta_n)$.

4. Results

In this section we present the comparison of SFA with three other popular data classification algorithms. The comparisons were made over seven real-world data sets taken from the UCI Machine Learning Repository [Asuncion and Newman (1998)] and one synthetic data set. We conclude that SFA had performance comparable to other standard classifiers with their control and kernel parameters optimized by an exhaustive grid search.

4.1. *Data sets*

We first compare prediction abilities of the classifiers on a synthetic data set and then on real-world data sets. An artificial binary Gaussian mixture toy data set was generated in the following manner. First ten means were generated from a bivariate Gaussian distribution $N((1, 0), \mathrm{I})$ and this class was labeled as "+1" and is shown by circles in Fig. 1. Similarly ten means from the distribution $N((0, 1), \mathrm{I})$ were generated and this class was labeled "−1" and is shown by squares in Fig. 1. The grey and white regions show classes −1 and +1 predicted by the classifier. One hundred observations belonging to each class were generated in the following way.

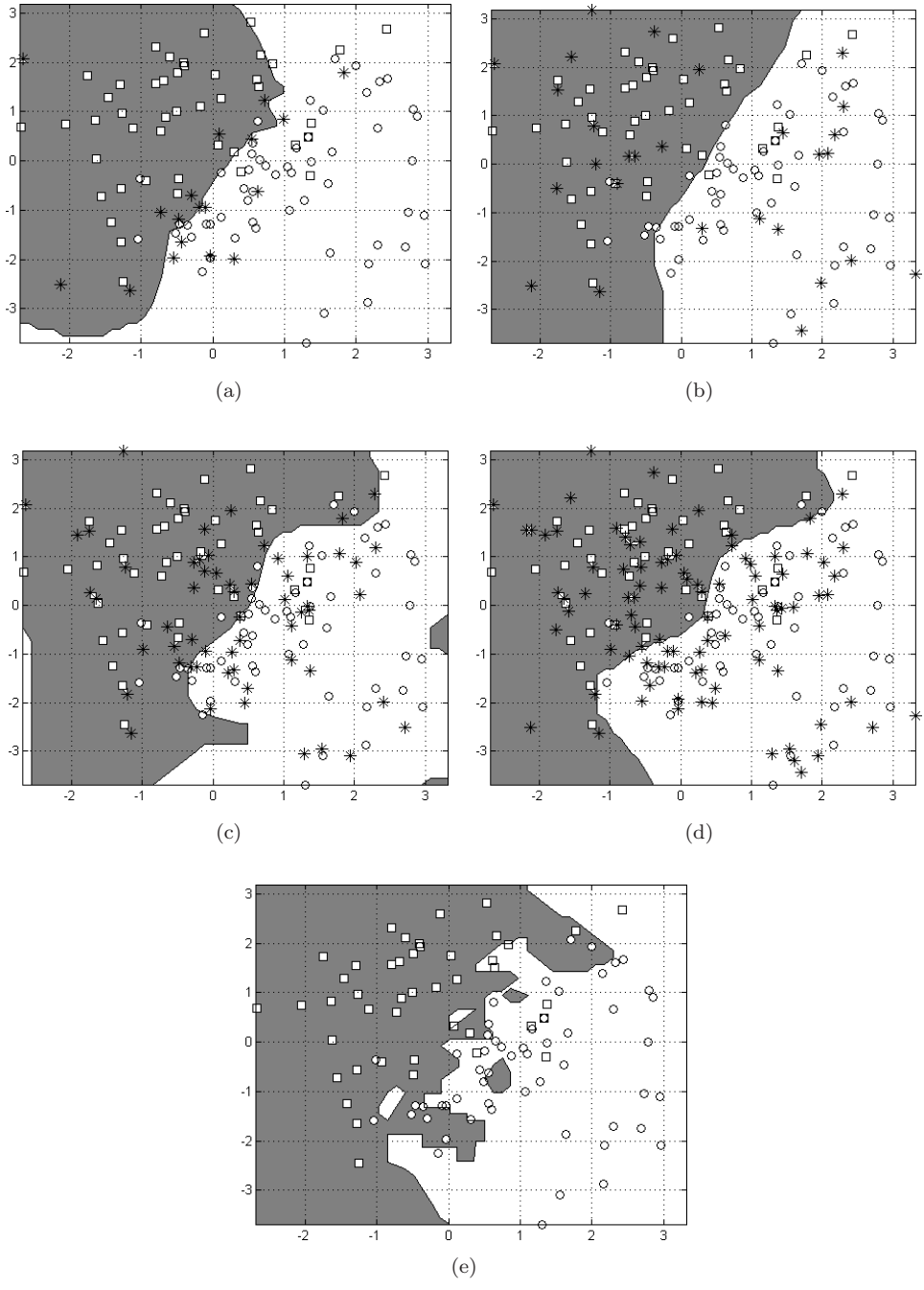

Fig. 1. Class decision boundaries produced by (a) SFA, (b) KMP (c) LS-SVM, (d) KMP-pre, and (e) K-NN. The hollow circles and squares represent the test points. Asterisks mark the support vectors used by the classifier and are not shown in 1(e) because k-NN does not use support vectors to construct its approximation. The grey and white regions show classes -1 and $+1$ predicted by the classifier.

Table 1. Specifications of the data sets used.

Data set	Number of input dimensions	Number of classes	Size of the data set
Toy	2	2	200
Iris	4	3	150
Wine	13	3	178
Glass	9	6	214
Housing	13	3	506
German Credit	24	2	1000
Balance Scale	4	3	625
Ionosphere	34	2	351

For each observation belonging to Class +1 a mean $m1$ was randomly selected from one of the ten means, which generated $N(m1, 0.5*\mathrm{I})$. Similarly for each observation belonging to Class −1 a mean $m2$ was randomly selected from one of the ten means which generated $N(m2, 0.7*\mathrm{I})$, thus producing a mixture of Gaussian clusters for each class.

Seven real-world application data sets were also included and downloaded from the UCI Machine Learning Repository. The HOUSING data set was converted to a 3-class problem for this study. The algorithms were compared on a total of eight data sets shown in Table 1.

4.2. *Experimental procedure*

The SFA classifier is compared to three other algorithms:

(1) **LS-SVM:** LS-SVMlab1.5 [Suykens (2002)], a Matlab/C toolbox was used to implement the Least Squares Support Vector Machines with Gaussian radial basis functions. It uses two user-controlled parameters: the regularization parameter (γ) and the width of the Gaussian radial basis function (σ).

(2) ***k*-NN:** SPIDER [Weston *et al.* (2006)], a Matlab-based machine learning toolbox was used to implement the k-nearest neighbor algorithm in this work. Gaussian radial basis functions were used to compute the distances between training/test points. It has two user-controlled parameters: the number of nearest neighbors (k) and the width of the Gaussian radial basis function (σ).

(3) **KMP:** Kernel Matching Pursuit [Vincent and Bengio (2002)] was also used as one of the algorithms for comparison. The performance of the basic version of KMP (**KMP**) and the pre-fitting version of KMP (**KMP-pre**) was investigated with Gaussian radial basis functions. In the implementation of KMP and KMP-pre a tolerance ($\tau = 1.0$) on the maximum absolute value of the residual was used to stop the addition of basis functions. This is the same stopping criterion as used in the implementation of SFA. Other than the tolerance both versions of the algorithm have one user-controlled parameter: the width of the Gaussian radial basis function (σ). Both versions were coded in Matlab version 7.5 to get the results presented in this paper.

The *one vs. all* scheme is used to combine different binary classifiers and perform multi-class classification for all the above-mentioned algorithms. All algorithms were implemented using the MATLAB programming environment on a Windows-configured PC with a Pentium 4 2.66 GHz processor and 1.0 GB of RAM.

4.2.1. *Grid search*

The parameters of LS-SVM, k-NN, KMP, and KMP-pre were determined by an exhaustive grid search. Both LS-SVM and k-NN have two control parameters so a two-dimensional grid was constructed between the following grid limits. For LS-SVM γ is chosen from the range 2^{-5} to 2^{12} and σ is chosen from the range 2^{-10} to 2^5. These grid intervals are taken from the technical report [Hsu *et al.* (2003)] where authors give recommendations about parameter grid search for support vector machines. For k-NN the grid search was performed over the two-dimensional parameter space (k, σ), k is chosen from the range 1 to 100 and σ is picked from the range same as chosen for LS-SVM. Each of the parameter ranges was discritized into ten equally spaced points to obtain a grid of 100 (γ, σ) and (k, σ) combinations. Once the training and test sets are obtained, they are loaded in the respective programs of the classifiers. Then training and testing were repeated on each grid point and classification accuracy was stored. Once this loose grid search is over a refined grid search is performed in the neighborhood of the obtained maximum classification accuracy. The above-mentioned two-dimensional grid is divided into four equal quadrants, and the quadrant in which the obtained maximum classification accuracy lies is determined. Then that quadrant is divided into 400 equally spaced grid points and training and testing are repeated on each grid point in that quadrant. For KMP and KMP-pre a line search was performed on 100 grid points equally spaced in the range 2^{-10} to 2^5. Consequently, optimal classification accuracy is obtained on the chosen test set for the above-mentioned grid intervals.

4.3. *Results*

First we compare the classifiers on the binary artificial data set described in Sec. 4.1. A data set with 200 points was generated and then was randomly partitioned into 100 training and 100 test points. Performance of the classifiers is measured by calculating the percentage accuracy as shown below:

$$\text{Percentage accuracy} = \left(\frac{Ntest - mtest}{Ntest}\right) * 100\%,$$

where $Ntest$ = number of points in the test set and $mtest$ = number of misclassifications. Percentage error of each classifier on the Toy data set is shown in Table 2.

The first column of the table shows the percentage classification accuracies of all classifiers on the Toy test set obtained by the grid search method mentioned in Sec. 4.2.1. The resulting parameters are given in the last two columns. The width parameter σ for SFA is not mentioned because SFA being adaptive in nature uses a

Table 2. Percentage accuracy of the classifiers on the Toy test data set with Gaussian basis functions using 50% of the total set for training. Dashes represent not applicable conditions.

Classifier	% accuracy	Sparsity	σ	Other parameters
LS-SVM	89.00	58	2.24	$\gamma = 1401.30$
KMP	88.00	100	1.23	–
KMP-pre	91.00	95	1.14	–
k-NN	81.00	–	28.58	$k = 1$
SFA	83.00	27	–	–

different width for each basis function. The classification accuracy of SFA is similar to other standard algorithms with their parameters optimized by an exhaustive grid search.

The second column of the table shows the number of training points that were used as basis functions. KMP used 100 basis functions but it used only 28 training points as basis centers, the rest of 72 basis functions were repetitions of the 28 already chosen basis functions. Since the width of all basis functions is the same choosing the same basis center means picking the same s-dimensional vector from the dictionary of all basis functions. KMP-pre on the other hand is an improvement over KMP since it does not pick an already chosen basis function. But it does not result in a sparse classifier choosing 95 of the 100 training points. Now it should be noted here that the sparsity of a greedy classifier is dependent on the stopping criterion. It is possible that choosing a different stopping criterion such as an upper bound on the support vectors, minimum description length criterion, Akaike information criterion, or a combination of different criteria can result in a sparser classifier. But in this work we chose a tolerance level on the maximum absolute value of the residual as the stopping criterion, as is usually done by signal processing algorithms. A comparison of the different stopping criterion for the proposed greedy algorithm deserves to be a part of a separate study. SFA used 27 basis functions to train on the Toy training set. Out of the 27 basis functions, seven of them were repetitions of already chosen basis centers. But since each basis function has a different width, repetition of a basis center does not mean that the same s-dimensional vector was picked from the dictionary of basis functions. LS-SVM used 58 training points as basis functions or support vectors. Figure 1 shows the classification boundaries obtained by the different classifiers on the Toy data set. Asterisks represent support vectors used by each classifier. Since k-NN does not involve the use of support vectors, asterisks are not shown in Fig. 1(e).

Since SFA, KMP, and KMP-pre are greedy methods the rate of convergence of $\langle \vec{r}_n, \vec{r}_n \rangle_D$ can be studied as the basis functions are added.

It can be seen from Fig. 2 that convergence of the residual error when SFA is used is faster than convergence rate of KMP or KMP-pre. All three algorithms start with the initial residual error of $\langle \vec{r}_n, \vec{r}_n \rangle_D = 100$. After the addition of the first basis function $\langle \vec{r}_n, \vec{r}_n \rangle_D = 84.70$ for KMP, $\langle \vec{r}_n, \vec{r}_n \rangle_D = 70.57$ for KMP-pre and $\langle \vec{r}_n, \vec{r}_n \rangle_D = 49.78$ for SFA. Also it should be noted that convergence rate of SFA on

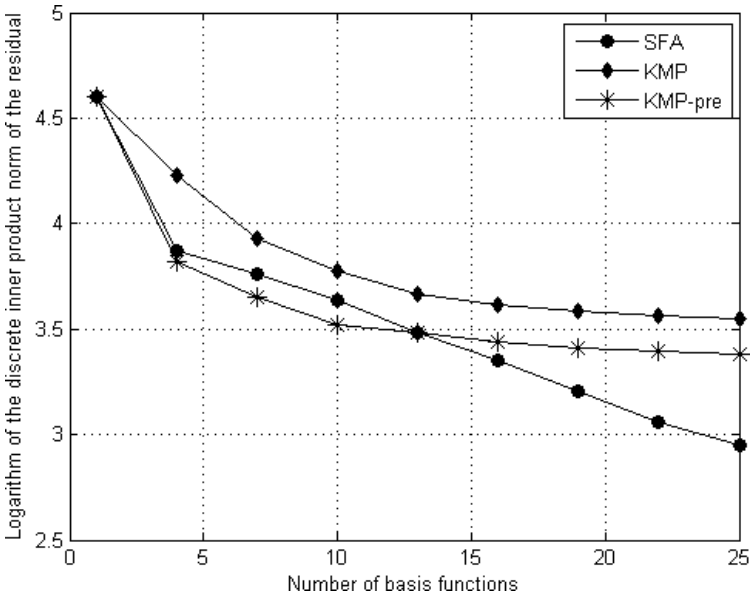

Fig. 2. Convergence of log $(\langle \vec{r}_n, \vec{r}_n \rangle_D)$ with the addition of the first 25 basis functions.

this problem is almost exponential. In Fig. 2 residual error convergence is shown up to 25 basis functions because all three algorithms used at least 25 basis functions to converge. SFA used only 27 basis functions before stopping, residual error by KMP-pre further decreases while the residual error almost remains constant using KMP.

4.4. *Classification data sets*

In this section compare the classifiers on seven real-world data sets taken from the UCI Machine Learning Repository. The training set for each classification data set constitutes of 50% of randomly chosen points and the test set contains the rest of the points. Kernel hyper-parameters for each algorithm, except SFA, are selected by an exhaustive grid search method described in Sec. 4.2.1. Table 3 shows the hyper-parameter values for each data set obtained after the grid search.

After obtaining the hyper-parameter values mean and standard deviation of the classification accuracies were estimated on 20 random training and test sets containing 50% of all the points in the data set. The percentage accuracies shown in Table 4 are unweighted in the sense that misclassification of one class is equally important as misclassification of any other class present in the data set.

4.5. *Sparsity*

In this work sparsity of a classifier is measured as the percentage of training data used by the classifier to predict the class of a test point. The sparsity of a classifier is

Table 3. Kernel hyper-parameter values of the classifiers obtained after an exhaustive grid search.

Data set	LS-SVM		k-NN		KMP	KMP-pre
	γ	σ	k	σ	σ	σ
Iris	107.82	8.00	9	22.63	1.40	2.56
Wine	431.19	10.46	15	27.39	4.02	32.00
Glass	323.40	9.93	6	25.01	0.53	0.70
Housing	107.82	12.38	1	5.96	2.79	9.88
German Credit	323.40	11.90	1	26.20	31.94	6.83
Balance Scale	215.61	10.86	14	23.82	3.25	1.73
Ionosphere	2048.00	19.25	1	32.00	1.81	1.44

Table 4. Percentage accuracy obtained by the algorithms using half of the data set to train and the other half to test the classifier using Gaussian basis functions. For each data set the highest mean accuracies are shown in bold.

Data set	LS-SVM	k-NN	KMP	KMP-pre	SFA
Iris	95.46 ± 1.88	**96.25 ± 1.43**	94.21 ± 3.98	91.58 ± 5.94	94.93 ± 2.87
Wine	**95.22 ± 2.11**	70.78 ± 3.46	71.61 ± 3.17	68.61 ± 2.99	70.22 ± 3.63
Glass	57.59 ± 3.80	62.13 ± 4.44	67.82 ± 3.39	66.99 ± 4.12	**68.80 ± 4.04**
Housing	**72.95 ± 1.99**	66.16 ± 2.05	66.24 ± 2.88	67.33 ± 2.84	68.44 ± 2.52
German Credit	**72.52 ± 1.46**	65.70 ± 1.10	70.79 ± 1.32	70.01 ± 1.99	69.87 ± 1.39
Balance Scale	84.30 ± 1.41	**88.85 ± 1.86**	88.43 ± 1.51	88.50 ± 1.14	86.92 ± 1.23
Ionosphere	**94.09 ± 1.55**	85.57 ± 3.35	89.60 ± 1.62	82.74 ± 3.06	83.60 ± 1.66

calculated as the average number of support vectors used by all the binary classifiers. The average is expressed as a percentage of the number of training points and the values are recorded in Table 5. It should be noted here that for all algorithms except SFA, the parameters of the classifier and the kernel hyper-parameters used are optimal in the sense that they give the maximum performance on the test set. Also classification of a test point by k-nearest neighbors uses the entire training set due to its inherent structure.

Table 5. Percentage of training points used for prediction of a test point. Values corresponding to the k-nearest neighbor classifier are not shown as it uses 100% of the training set to predict the label of a test point. Sparsest results are shown in bold.

Data set	LS-SVM	KMP	KMP-pre	SFA
Iris	54.71	37.76	17.59	**8.90**
Wine	69.61	88.50	34.69	**20.27**
Glass	60.42	69.68	47.46	**16.60**
Housing	63.60	95.12	83.78	**23.77**
German Credit	93.60	100.00	75.59	**30.40**
Balance Scale	56.67	99.68	36.63	**11.50**
Ionosphere	79.49	100.00	43.44	**20.20**

5. Conclusions and Future Avenues of Research

A sparse approximation algorithm is developed which constructs the approximation in a greedy fashion. The main advantage of this algorithm over any other routine used for classification purposes is that it determines all parameters during learning and does not involve any user interaction. It assures the user that it is the optimum performance given by the algorithm, while any other algorithm cannot give this assurance to the user. Experimental tests were conducted over a number of real-world application data sets and its performance was compared with other standard classification routines. Experiments on both artificial and real-world data sets show that SFA gave sparser approximations with comparable prediction accuracy.

Best possible approximation to a problem can be constructed by solving a three-dimensional optimization problem for the coefficients (c_n, b_n), the width parameter (σ_n), and the center of the basis function (x_n^*) at each stage. Incorporating the method of weighted residuals we calculated the optimum value of the coefficients which leaves us with a two-dimensional optimization problem for the center and the widths at each stage. In this algorithm we used a heuristic given in Sec. 3.1 to locate the center of the radial basis function at each stage instead of optimizing it to reduce computational time and expense. Then we are left with a one-dimensional optimization problem for the width of the basis function which we solve at each stage in the algorithm. This algorithm gives a suboptimal rate of convergence for the approximation error. Research is in progress to mathematically formulate the dependence of the residual as a function of the number of basis functions. We believe that this formulation would result in the optimal value of the width parameter at each stage which could then be used to determine the optimum center of the radial basis function at each stage. An algorithm like this would reduce the number of optimizations to three to four thus accelerating the computational time taken.

Research is in progress to formulate the estimation error of the algorithm. Having obtained convergence rates for approximation and estimation error we can bound the generalization error of the algorithm using radial basis function. This research is being done for radial basis functions but in general SFA could also be used with other standard basis functions like the sigmoid and the hyperbolic tangent.

After obtaining generalization error bounds of SFA, it is envisioned as a tool to work with the experimentalists to better understand the design and working of a system. SFA cannot only be used as a pattern recognition tool given a data set, but also help the experimentalists accelerate through the test matrix by estimating how many more number of experiments need to be conducted for a user defined accuracy level. Being greedy in nature, ideas of active learning [Niyogi (1995)] can be incorporated with filter or wrapper feature selection methods to better design of the test matrix.

Acknowledgments

Support for this work was provided by the NASA Ames Research grant NCC-2-8077 and NASA Cooperative Agreement No. NCC-1-02038.

References

Asuncion, A. and Newman, D. J. (1998). UCI Repository of machine learning databases, Available electronically at http://archive.ics.uci.edu/ml/.

Barron, A. R. (1993). Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Trans. Inform. Theory 39*, **3**: 930–945.

Breiman, L. *et al.* (1984). *Classification and Regression Trees.* CRC Press.

Buhmann, M. D. (2003). *Radial Basis Functions.* Cambridge University Press, New York.

Chen, S. S., Cowan, C. F. N. and Grant, P. M. (1991). Orthogonal least squares learning for radial basis function networks. *IEEE Trans. Neural Networks*, **2**: 302–309.

Chen, S. S. (1995). *Basis Pursuit.* Ph.D. Thesis, Stanford Uni., California.

Couvreur, C. and Bresler, Y. (1999). On the optimality of the backward greedy algorithm for the subset selection program. *SIAM J Matrix Anal. Appl. 21*, **3**: 797–808.

Cun, Y. L. *et al.* (1990). Handwritten digit recognition with a back-propagation network. *Adv. Neural Inform. Process. Syst.*, **2**: 396–404.

Dasarathy, B. V. (1991). *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques.* IEEE Comp. Soc. Press, Los Alamitos.

Fahlman, S. E. and Labiere, C. (1990). The cascade-correlation learning architecture. *Adv. Neural Inform. Process. Syst.*, **2**: 524–532.

Fletcher, C. A. J. (1984). *Computational Galerkin Methods.* Springer-Verlag, New York.

Freund, Y. and Schapire, R. E. (1996). Experiments with a new boosting algorithm. *Proc. Intern. Conf. Machine Learning*, **13**: 148–156.

Friedman, J. H. and Stuetzle, W. (1984). Projection pursuit regression. *J. Amer. Stat. Assoc.*, **79**: 599–608.

Girosi, F. (1998). An equivalence between sparse approximation and support vector machines. *Neural Comput.*, **10**: 1455–1480.

Grapel, T., Herbrich, R. and Shawe-Taylor, J. (2000). Generalization error bounds for sparse linear classifiers. *Ann. Conf. Comput. Learn. Theory*, **13**: 298–303.

Hsu, C. W., Chang, C. C. and Lin, C. J. (2003). A practical guide to support vector classification. *Tech. Rep.*, Dept. of Comp. Sci. and Info. Eng. Natl. Taiwan Uni., Taipei.

Jones, L. K. (1990). Constructive approximations for neural networks by sigmoidal functions. *Proc. IEEE 78*, **10**: 1586–1589.

Jones, L. K. (1992). A simple lemma on greedy approximation in Hilbert space and convergence rates for projection pursuit regression and neural network training. *The Ann. Stat.* 20, **1**: 608–613.

Mallat, S. G. and Zhang, Z. (1993). Matching pursuits with time-frequency dictionaries. *IEEE Trans. Signal Process*, *41*, **12**: 3397–3415.

Nair, P. B., Choudhary, A. and Keane, A. J. (2002). Some greedy learning algorithms for sparse regression and classification with mercer kernels. *J. Mach. Learn. Res.*, **3**: 781–801.

Natarajan, B. K. (1995). Sparse approximate solutions to linear systems. *SIAM J. Comp. 25*, **2**: 227–234.

Niyogi, P. (1995). *Active Learning by Sequential Optimal Recovery.* AI Memo-1514, Arti. Intell. Lab., Massachusetts Inst. of Technology.

Rifkin, R. and Klautau, A. (2004). In defense of one-vs-all classification. *J. Mach. Learn. Res.*, **5**: 101–141.

Scholkopf, B. and Smola, A. J. (2002). *Learning with Kernels*. The MIT Press, Cambridge, Massachusetts.

Suykens, J. A. K. *et al.* (2002). *Least Squares Support Vector Machines*. World Scientific, Singapore.

Vincent, P. and Bengio, Y. (2002). Kernel matching pursuit. *Mach. Learn.*, **48**: 165–187.

Wang, Z. *et al.* (1994). A procedure for determining the topology of multilayer feedforward neural networks. *Neural Networks* 7, **2**: 291–300.

Weston, J. *et al.* (2006). Software available electronically at http://www.kyb.mpg.de/bs/people/spider/main.html. Max Planck Institute for Biological Cybernetics.

Yingwei, L., Sundararajan, N. and Saratchandran, P. (1997). A sequential learning scheme for function approximation using minimal radial basis function neural networks. *Neural Comput.* **9**, 2: 461–478.