International Journal of Pattern Recognition
and Artificial Intelligence
Vol. 21, No. 7 (2007) 1–24

World Scientific
www.worldscientific.com

FEATURE EXTRACTION FOR CLASSIFICATION USING STATISTICAL NETWORKS

ANIL KUMAR GHOSH

Centre for Mathematics and Its Applications, Mathematical Sciences Institute
Australian National University, Canberra, ACT 0200, Australia
anilkghosh@rediffmail.com

SMARAJIT BOSE

Theoretical Statistics and Mathematics Unit, Indian Statistical Institute
203, Barrackpore Trunk Road, Kolkata 700108, India
smarajit@isical.ac.in

In a classification problem, quite often the dimension of the measurement vector is large. Some of these measurements may not be important for separating the classes. Removal of these measurement variables not only reduces the computational cost but also leads to better understanding of class separability. There are some methods in the existing literature for reducing the dimensionality of a classification problem without losing much of the separability information. These dimension reduction procedures usually work well for linear classifiers. In the case where competing classes are not linearly separable, one has to look for ideal "features" which could be some transformations of one or more measurements. In this paper, we have tried to tackle both the problems of dimension reduction and feature extraction by considering a projection pursuit regression model. The single hidden layer perceptron model and some other popular models can be viewed as special cases of this model. An iterative algorithm based on backfitting is proposed to select the features dynamically and cross-validation is used to select the ideal number of features. We carry out an extensive simulation study to show the effectiveness of this fully automatic method.

Keywords: Artificial neural networks; backfitting; classification using splines; cross-validation; feature selection; projection pursuit regression.

1. Introduction

In high-dimensional classification problems, it is often seen that only a smaller number of features (some functions of original variables) contain most of the information for separating the classes. It is therefore desirable to identify these features so that it helps to reduce the dimensionality of the problem which in turn facilitates the visualization of class separability in a lower dimensional space and at the same time reduces the computational and storage costs substantially.

If we restrict ourselves only to linear classifier, results for feature extraction are available in the literature (see e.g. Refs. 14 and 24). However, in practice, linear classifiers are often found to be inadequate. Therefore, extraction of important features for nonlinear classification is an important statistical problem. For linear classifiers the features are typically linear combinations of the original variables, which we will consider as linear features in this article. For nonlinear classification, in addition, we consider features which are nonlinear transformations either of original variables or of linear features. We will consider these as nonlinear features.

In regression, projection pursuit regression model (see e.g. Refs. 16 and 26) was developed to compensate for this inadequacy, and artificial neural network models[8,34,35] provide much more flexibility for classification. Both have their own strengths and limitations. In this paper, we try to combine these two approaches so that we can develop a method which is more flexible and statistically meaningful as well. It is shown later that many of the existing models are special cases of the resulting model. We propose an iterative feature selection (IFS) algorithm for dynamic adjustment of features and cross-validation techniques are used for selection of the final model appropriate to a specific problem.

The organization of the paper is as follows. The problem of classification is briefly described and the available tools for solving these problems are reviewed in Sec. 2. In Sec. 3, we discuss the motivation for the new method and compare the corresponding model with the existing ones. The algorithm is described in Sec. 4. Experimental results illustrating the new method is presented in Sec. 5 where we use some simulated and benchmark data sets to compare the method with some existing classification techniques. Finally, a summary and concluding remarks are given in Sec. 6.

2. Classification Techniques

In classification problems, one aims to develop a decision rule $d(\mathbf{x}) : R^d \to \{1, 2, \ldots, J\}$ based on the available training sample $\{(\mathbf{x}_n, c_n) : \mathbf{x}_n \in R^p,\ c_n \in \{1, 2, \ldots, J\};\ n = 1, 2, \ldots, N\}$ for classifying multivariate observations $\mathbf{x}$ into one of J competing populations. Bayes rule (see e.g. Ref. 1) assigns an observation $\mathbf{x}$ to the class with the largest conditional probability $p(j|\mathbf{x})$. An accurate approximation of these unknown probabilities helps us to formulate a decision rule capable of achieving nearly the optimal misclassification rate. These probabilities can be estimated either parametrically or nonparametrically. In parametric approaches (see e.g. Refs. 1, 19 and 31), the measurement vector $\mathbf{x}$ is assumed to have a known distribution with unknown parameters. For instance, in Fisher's (see Ref. 15) linear or quadratic discriminant analysis (LDA and QDA), this known distribution is taken as multivariate normal with different parameters for different classes. Performance of a parametric classifier depends heavily on the validity of parametric model assumptions, and when one or more of these conventional assumptions are violated, parametric classifiers often fail to adequately approximate the class

boundaries. In such situations, one usually prefers nonparametric methods, which are more flexible and robust. Notably nonparametric methods like classification trees (see e.g. Ref. 5), flexible discriminant analysis (see e.g. Ref. 23), nearest neighbors (see e.g. Refs. 10 and 18), kernel discriminant analysis (see e.g. Refs. 13 and 22), neural networks and support vector machines (see e.g. Refs. 11, 37 and 42) have been shown to outperform the parametric approaches in a wide variety of problems.

Many of these classification techniques including the neural network method regress indicator variables $Y_1, \ldots, Y_J$ (defined for J competing classes) on the measurement space to estimate $E(Y_j|\mathbf{x}) = p(j|\mathbf{x})$ $(j = 1, 2, \ldots, J)$, the posterior class probabilities. In particular, Bose[3] followed this principle to develop Classification Using Splines (CUS), which is based on nonparametric additive regression. This method essentially considers smooth functions of the original variables as new features and considers a linear model of these features to estimate posterior class probabilities. The proposed method in this paper can be viewed as a more generalized version of CUS.

3. Features for Nonlinear Classification

In classification problems, when class boundaries are highly nonlinear, one popular practice is to find suitable nonlinear features such that the competing classes are linearly separable in that feature space. Linear classification in that feature space essentially leads to a nonlinear separation in the space of the original measurement variables. Well-known methods like support vector machines and kernel fisher discriminants (see e.g. Ref. 32) adopt these techniques and use some suitable nonlinear kernel functions for nonlinear classification. Ghosh and Chaudhuri[21] also used a similar strategy for depth based nonlinear classification. However, instead of estimating features from the data, they worked with some predefined features. In a recent work, Zhu[43] tried to extract important features from the data by maximizing a likelihood ratio criterion. Neural network training algorithms also try to extract ideal features from the data automatically, but this popular method lacks meaningful statistical interpretations.

As we have mentioned in the previous section, CUS considers additive models (see e.g. Ref. 7) to estimate the posterior probabilities, and the estimates are of the form

$$\widehat{p}(j|\mathbf{x}) = \phi_{j0} + \sum_{i=1}^{p} \phi_{ji}(x_i), \quad j = 1, 2, \ldots, J, \tag{1}$$

where ϕ_{j0} is a constant and ϕ_{ji} are smooth univariate functions. Therefore, CUS may not perform well when the class boundaries cannot be approximated well by additive models.

Method of successive projection (see Ref. 4) was proposed to compensate for this inadequacy. It uses the estimated posterior probabilities obtained by CUS as the new measurement variables and repeats the CUS procedure with them to get

the final probability estimates

$$\widehat{p}(j|\mathbf{x}) = \phi_{j0}^{(2)} + \sum_{k=1}^{J-1} \phi_{jk}^{(2)} \left(\phi_{k0}^{(1)} + \sum_{i=1}^{p} \phi_{ki}^{(1)}(x_i) \right), \quad j = 1, 2, \ldots, J, \tag{2}$$

where $\phi^{(1)}$ and $\phi^{(2)}$ are smooth univariate functions estimated in the first and second iterations, respectively.

In the process, interactions are introduced indirectly in the model much in the same manner as in the multilayer perceptron model. Successive projection has two basic steps, one for variable transformation and the other for probability estimation. This is similar to the Multilayer Perceptron (MLP) model with a single hidden layer where posterior estimates are of the form:

$$\widehat{p}(j|\mathbf{x}) = \rho^{(2)}(\theta_j^{(2)} + \sum_k w_{jk}^{(2)} \rho^{(1)} \left(\theta_k^{(1)} + \sum_m w_{mk}^{(1)} x_m \right), \tag{3}$$

where $\theta^{(r)}$, $w^{(r)}$ and $\rho^{(r)}$ denote bias terms, weight functions and transformation functions associated with the rth $(r = 1, 2)$ layer. Several choices for the transformation function ρ is available in the literature. Because of its ability for uniform approximation of any continuous function (see e.g. Ref. 25) on a compact support, sigmoidal function $\sigma(t) = 1/(1 + e^{-t})$ is one of the most popular choices.

However, one disadvantage of successive projection is that the transformations are estimated once and for all. It lacks the flexibility of neural networks where features are modified iteratively. As a result, when the posterior probability estimates obtained by CUS are not good, the improvement over CUS is often not that significant.

The proposed iterative feature selection (IFS) algorithm, instead of using CUS, in the first step, tries to extract the best possible linear features (linear combinations) of the form $\boldsymbol{\alpha}_i'\mathbf{x}$ $(i = 1, 2, \ldots, p)$ from the data, and then performs CUS with them to estimate the posterior class probabilities. Backfitting is used for dynamic adjustment of $\boldsymbol{\alpha}_i$ and that makes the method very flexible. Like neural networks, IFS can adjust the linear combinations $\boldsymbol{\alpha}_i'\mathbf{x}$ iteratively to improve its performance. It can be viewed as an approach to estimate the posterior probabilities $p(j|\mathbf{x})$, $(j = 1, 2 \ldots, J)$ by projection pursuit regression models

$$\widehat{p}(j|\mathbf{x}) = \psi_j(\mathbf{x}) = \phi_{j0} + \sum_{i=1}^{p} \phi_{ji}(\boldsymbol{\alpha}_i'\mathbf{x}). \tag{4}$$

In projection pursuit, one has to find the ideal linear combinations and the corresponding transformation functions as well. IFS tries to accomplish that task automatically, and the resulting decision rule seems to provide a much better improvement over CUS, much more than the method of successive projections.

The model used in IFS has some similarities with some of the existing classification methods. Obviously, the additive model used in CUS is a special case of IFS when $\boldsymbol{\alpha}_i$s are unit vectors along the co-ordinate axes. In fact, if sigmoidal or

other known transformations are used as the smooth univariate functions ϕ_{ji}, this model turns out to be a perceptron model with one hidden layer. In the perceptron model, a fixed set of transformation functions (which could possibly be different) is used, and a given function is approximated to any level of accuracy by increasing the number of hidden nodes. This implies more number of linear features (linear combinations) and hence a transformation to a high-dimensional feature space from the original measurement space.

Since our aim is to reduce the dimensionality of the problem, in IFS, we restrict the number of features to the dimension of the original measurement vector and try to approximate the transfer functions by using additive splines. Therefore, not only the linear combinations are estimated from the data iteratively but so are the transfer functions. The fewer number of linear combinations may help to identify simpler structures in the data which has been illustrated with the simulated Example 1 in Sec. 5.1.

Use of splines as transformation functions makes the posterior estimates affine-invariant, and it also leads to the universal approximation property like the sigmoidal function. Given the set of linear features, using sufficiently many knots, splines can approximate any continuous function on that space to any desired level of accuracy (see Ref. 3). A detailed discussion is given in Sec. 4.5. The IFS algorithm actually evolved from a similar scheme applied for training perceptrons earlier which resulted in Backfitting Neural Networks (see Ref. 20). Mackay[29] has also proposed many modifications to the traditional backpropagation training algorithm including regularizations, model selections but he considered a different Bayesian approach.

4. Description of IFS Algorithm

IFS projects the posterior probabilities $p(j|\mathbf{x})$, $(j = 1, 2, \ldots, J)$ into the additive class of functions spanned by the cubic spline functions of $\boldsymbol{\alpha}_i'\mathbf{x}$ $(i = 1, 2, \ldots, p)$. Hence $\widehat{p}(j|\mathbf{x})$ is of the form:

$$\widehat{p}(j|\mathbf{x}) = \psi_j(\mathbf{x}) = \phi_{j0} + \sum_{i=1}^{p} \phi_{ji}(\boldsymbol{\alpha}_i'\mathbf{x}), \quad \text{where } ||\boldsymbol{\alpha}_i|| = 1 \quad \forall\, i = 1, 2, \ldots, p. \tag{5}$$

The constants ϕ_{j0}, direction vectors $\boldsymbol{\alpha}_i$ and the transformation functions (cubic splines) ϕ_{ji} $(j = 1, 2, \ldots, J;\ i = 1, 2, \ldots, p)$ are determined iteratively. Cubic splines are pieces of cubic polynomials joined together at the knots suitably placed on the range of a variable. They are sufficiently smooth, and have continuous derivatives up to the second order. Following the suggestion of Breiman,[7] we also put restrictions on splines to make the extrapolated fit linear in the tails for reducing the effect of high variability at the end points. IFS starts with sufficiently many knots and then at the end of the iterative learning procedure, it reaches a smaller dimensionality by the method of backward deletion. The optimum dimensionality is selected by cross-validation. Guidelines for other related issues like selection of basis functions,

method of backward deletion are as given in Ref. 7 which we discuss briefly in this section.

4.1. *Selection of basis functions and parameters*

For data with univariate $Z_n = \boldsymbol{\alpha}'\mathbf{x}_n$, $(n = 1, 2, \ldots, N)$ and K knots $t_1 < t_2 < \cdots < t_K$ in the range $[Z_{(1)} = \min_n Z_n, Z_{(N)} = \max_n Z_n]$, a cubic spline ϕ is a cubic polynomial in each interval (t_k, t_{k+1}), for $k = 1, 2, \ldots, K-1$, and it has continuous derivatives up to the second order. It is easy to observe that for a given set of K knots, the space of cubic splines is $(K+4)$ dimensional. Two sets of functions commonly used as the bases for the class of cubic splines are the power basis and the B-spline basis (see e.g. Ref. 12). In our algorithm, we use the former because deletion of knots is computationally more feasible with this basis. The power basis for univariate splines consists of the functions $1, Z, Z^2, Z^3, (Z-t_k)_+{}^3,\ k = 1, 2, \ldots, K$, where $Z_+ = Z$ if $Z > 0$ and 0 otherwise. To make the extrapolated spline fit outside the range $[Z_{(1)}, Z_{(N)}]$ linear, Breiman[7] suggested to impose the conditions $\phi''(Z_{(1)}) = \phi'''(Z_{(1)}-) = 0$ for the left tail, and $\phi''(Z_{(N)}) = \phi'''(Z_{(N)}+) = 0$ for the right tail.

A convenient power basis for this space can be constructed by starting with the functions $1, Z, (Z-t_k)_+^3,\ k = 1, 2, \ldots, K$. Therefore, the restricted spline function is of the form

$$\phi(Z) = \theta + \gamma Z + \sum_{k=1}^{K} \beta_k (Z - t_k)_+^3, \tag{6}$$

which automatically satisfies the condition for the left tail. To make $\phi(Z)$ linear to the right of $Z_{(N)}$, Breiman[7] suggested to take $t_1 = Z_{(1)}$ and an additional knot t_{K+1} at $Z_{(N)}$ to cancel out the higher order terms. Condition for linearity at the right end imposes the constraints

$$\sum_{k=1}^{K+1} \beta_k = 0 \quad \text{and} \quad \sum_{k=1}^{K+1} \beta_k (Z - t_k) = 0 \quad \text{for} \quad Z \geq Z_{(N)}. \tag{7}$$

For multivariate $\mathbf{Z} = (Z_1 = \boldsymbol{\alpha}_1'\mathbf{x}, Z_2 = \boldsymbol{\alpha}_2'\mathbf{x}, \ldots, Z_p = \boldsymbol{\alpha}_p'\mathbf{x})$, we place $(K+1)$ knots on each Z-coordinate and use the power basis, which is the collection of the functions $1, Z_i, (Z_i - t_{i_k})_+^3$, $(k = 1, 2, \ldots, (K+1);\ i = 1, 2, \ldots, p)$. Conditions for linearity are imposed similarly on each of the p coordinates.

4.2. *The algorithm*

Different steps of the IFS algorithm are as follows:

Step 1. Indicator variables $Y_1, Y_2, \ldots, Y_J$ are defined for J competing classes.

Step 2. Direction vectors $\boldsymbol{\alpha}_i$ $(||\boldsymbol{\alpha}_i|| = 1)$ $(i = 1, 2, \ldots, p)$ are initialized and the linear features (combinations) $Z_1, Z_2, \ldots, Z_p$ are computed.

$$Z_i = \boldsymbol{\alpha}_i'\mathbf{x}, \quad i = 1, 2, \ldots, p. \tag{8}$$

Step 3. For each Z_i $(i = 1, 2, \ldots, p)$, K knots $t_{i_1} < t_{i_2} < \cdots < t_{i_K}$ are placed in the range $[\min_n(Z_{i_n}), \max_n(Z_{i_n})]$. Basis functions are computed (as described in Sec. 4.1).

Step 4. Y_j is regressed on basis functions to get the initial estimates for $\phi_{j0},\ \phi_{j1}, \ldots, \phi_{jp},\ (j = 1, 2, \ldots, J)$. Posterior probability estimates $(\widehat{Y}_{j_n} = \psi_j(\mathbf{x}_n))$ and residuals $(r_{j_n} = Y_{j_n} - \psi_j(\mathbf{x}_n))$, $(j = 1, 2, \ldots, J,\ n = 1, 2, \ldots, N)$ are computed to find the residual sum of squares (RSS) $= \sum_{j=1}^{J} \sum_{n=1}^{N} r_{j_n}^2$.

Step 5. Backfitting is used to readjust $Z_1, Z_2, \ldots, Z_p$. At any stage $\boldsymbol{\alpha}_1$ is adjusted by a factor $\boldsymbol{\delta}_1$, where $\boldsymbol{\delta}_1$ is the least squares solution obtained by minimizing

$$\text{RSS} \simeq \sum_{j=1}^{J} \sum_{n=1}^{N} [r_{j_n} - \boldsymbol{\delta}_1' \eta_{j_n}^{(1)}]^2, \tag{9}$$

where $\eta_{j_n}^{(1)} = \frac{\partial \phi_{j1}}{\partial \boldsymbol{\alpha}_1} | \mathbf{x} = \mathbf{x}_n$ is computed at the current estimates of $\boldsymbol{\alpha}_1$ and ϕ_{j1}. This new estimate of $\boldsymbol{\alpha}_1$ is normalized, and Z_1 is recomputed. Knots are placed on the range of Z_1 to find the new basis functions. $Y_j - \sum_{i \neq 1} \phi_{ji}(Z_i)$ are regressed on those basis functions to estimate ϕ_{j0} and ϕ_{j1} $(j = 1, 2, \ldots, J)$. However, these adjustments are made only when this new estimate of $\boldsymbol{\alpha}_1$ reduces the current value of RSS, and in that case, probability estimates and residuals are also readjusted. Otherwise, all the estimates, knots and basis functions are kept unchanged. After $\boldsymbol{\alpha}_1$, we proceed in the similar way to adjust $\boldsymbol{\alpha}_2$ and all other directions. Thus, $Z_1, Z_2, \ldots, Z_p$ are modified one by one. This step is repeated until no significant improvement is observed in RSS.

Step 6. Y_j is regressed on current basis functions to get new estimates for $\phi_{j0}, \phi_{j1}, \ldots, \phi_{jp},\ (j = 1, 2, \ldots, J)$. $\widehat{Y}_{j_n}$ and r_{j_n} $(j = 1, 2, \ldots, J;\ n = 1, 2, \ldots, N)$ are adjusted accordingly.

The last two steps (Steps 5 and 6) are repeated until convergence is achieved (relative reduction in RSS is very small over a number of consecutive iterations). A detailed version of this algorithm is given in the Appendix.

Like CUS (see Ref. 3), IFS puts no restriction on ψ_j $(j = 1, 2, \ldots, J)$ to ensure that they are in $[0, 1]$. Imposing positivity restrictions by any manner results in much more complicated but not necessarily better classification (see Ref. 27). Due to special type of effect of bias-variance decomposition on misclassification rates (see e.g. Ref. 17), IFS leads to fairly good results in spite of having posterior estimates outside the $[0, 1]$ range. However, inclusion of the intercept term in the set of basis functions, ϕ_{j0} $(j = 1, 2, \ldots, J)$ guarantees the additivity constraint $(\sum_{j=1}^{J} \widehat{\psi}_j(\mathbf{x}_n) = 1$, $\forall n = 1, 2, \ldots, N)$.

4.3. *The backward deletion procedure: Selection of optimum dimensionality*

As we have mentioned before, IFS algorithm starts with a reasonably large number of initial knots. After fitting this full model, backward deletion is used to

attain models with smaller dimensionalities. At each step, we consider each of the undeleted knots as possible candidates for deletion, and delete that one which leads to least increase in the training set RSS. To maintain linearity at the left tail, linear basis functions cannot be considered as a candidate for deletion until all the knots corresponding to that variable have been deleted. During the deletion process, some restrictions are imposed to maintain the linearity of the fitted functions beyond the rightmost undeleted knots. If $t_{i_{(K+1)}}$, the rightmost knot on Z_i, is deleted, the condition of linearity to the right of t_{i_K} can be imposed by adding another constraint $\sum_{k=1}^{K} \beta_{i_k} = 0$. One needs to solve the least squares problem under one or two such constraints which are to be satisfied for each of the Z-variables. The whole deletion procedure is done by modified Gaussian sweep algorithm (see e.g. Ref. 7).

Backward deletion generates a sequence of nested models indexed by the dimensionality of the constrained space. The problem then reduces to selection of an optimum one under some criterion. As the optimum dimensionality depends on the problem itself, it is desirable to find it using the training set observations. If resubstitution error rate is used as a criterion, naturally one of the larger models would be selected which may not be the best one for classifying future observations. Therefore, to arrive at a parsimonious model, we adopt the cost complexity criterion. Cost complexity for a model with dimension i is defined as

$$R_\gamma(i) = R(i) + \gamma i, \quad i = 0, 1, 2, \ldots \tag{10}$$

where $R(i)$ is the training set misclassification rate for the corresponding model, and γ is the cost complexity parameter. Clearly, $\gamma = 0$ leads to resubstitution error rate criterion when the cost complexity gets minimized by one of the larger models. This model remains optimum up to a certain positive value of γ after which another smaller model turns out to be the cost minimizer. In the process, a number of intervals and the corresponding minimizing models are obtained, and the number of competitive models usually gets reduced. Suppose we get m such intervals for γ $\{(\gamma_{t-1}, \gamma_t);\ t = 1, 2, \ldots, m\}$, and the corresponding models are $M_1 \succ M_2 \succ \cdots \succ M_m$. The problem of selection of final model then reduces to selection of an ideal cost parameter for which we use cross-validation.

In V-fold cross-validation (see e.g. Refs. 35 and 41), stratified random sampling is carried out to divide the whole training set into V groups $L_1, L_2, \ldots, L_V$ of sizes as nearly equal as possible. Observations belonging to different classes are used as different strata. Leaving one group $L_r\,(r = 1, 2, \ldots, V)$ at a time as a hold-out sample, IFS is used on the remaining observations. We start with the same number of initial knots per variable that was used with the whole training set, and then backward deletion is carried out in the same way. In the process, a nested sequence of models (as discussed above) is generated, and each time the resubstitution error rate is computed. The cost function is then minimized over these models for different γ, more specifically for $\gamma_t^* = \sqrt{\gamma_{t-1}\gamma_t},\ (t = 1, 2, \ldots, m)$. Let $M_t^{(r)}$ be the models which minimize the cost functions $R_{\gamma_t^*}\,(t = 1, 2, \ldots, m)$.

These models are then used to classify the observations in the hold out sample (L_r), and in each case the number of misclassifications is computed. We repeat the same procedure over the V groups to estimate the misclassification error rates (Δ) for different γ_t^*.

$$\widehat{\Delta}(\gamma_t^*) = \frac{1}{N}\sum_{r=1}^{V}\sum_{\{n:(\mathbf{x}_n,c_n)\in L_r\}} I(d_t^{(r)}(\mathbf{x}_n) \neq c_n), \quad t = 1, 2, \ldots, m, \tag{11}$$

where $d_t^{(r)}$ is decision rule obtained from the model $M_t^{(r)}$. The value of γ_t^* that minimizes $\widehat{\Delta}$, is used as the ideal value of the cost complexity parameter, and the corresponding model is selected as the final model. In our algorithm, we use $V = 10$ for cross-validation. Further details on cost-complexity pruning is available in Ref. 5.

4.4. *Number and placement of initial knots: Selection of initial features*

Number of initial knots should be chosen carefully. To explain local patterns of the measurement space, this number should be reasonably large. From our simulation study, we feel that the result is not too sensitive as long as reasonably large number of knots are used. Our experience suggests that 10–12 knots per variable are enough for a moderately large sample size. Cross-validation can also be used to find this number. However, one must be careful when working with a large number of knots. Not only it leads to higher variability in the model selected but may also lead to a matrix $Z'Z$ which cannot be stably inverted. This problem of singularity can be tackled by carefully placing the knots. Equispaced knots in the range works well in many cases but performs poorly when Z_is have large gaps between some consecutive observations. Knot placement based on order statistics seems to be a better choice.

Initial direction vectors $\boldsymbol{\alpha}_i^0$ $(i = 1, 2, \ldots, p)$ have to be specified as well. From our empirical experience we feel that instead of starting randomly, it is usually better to start with the linear features that maximize the linear separation among the classes. These direction vectors are the eigen vectors of $W^{-1}(W + B)$, where B and W stand for between class and within class sum of square matrices. Of course one can also start with the CUS model where $\boldsymbol{\alpha}_i^0$s are taken as the unit vectors along the co-ordinate axes (i.e. the original variables as the initial linear features). In the results reported in Sec. 5, IFS (features) and IFS (variables) represent the results obtained using these two different starting points, respectively.

4.5. *Convergence issues*

There are essentially three minimization procedures in IFS. Minimization over the linear features (the α parameters) which is done iteratively, minimization over the class of additive spline functions which is done by estimating the β parameters using least squares and minimization over the number of linear features which is

done by cost-complexity cross-validation. While convergence of the entire algorithm requires rather involved mathematical exercise, results regarding each of the three minimization are available in the literature which provides justification for the effectiveness of the algorithm.

The minimization over the α parameters is done by backfitting which was effectively used by Breiman and Friedman.[6] For estimating the additive transformations of several explanatory variables, they considered stepwise minimization of the error sum of squares involving the transformation function of only one variable at a time, and repeating the procedure until convergence. The advantage is that at each step, the error sum of squares is reduced by least squares which guarantees convergence. However there may not be a unique minimum for this part of the optimization.

Given the linear features, the IFS procedure is essentially the CUS procedure on the transformed feature space instead of the original measurement space. Convergence of IFS has been proved in Ref. 2. The proof relies on the universal approximation property of cubic splines.

Finally the optimality of the cost-complexity cross-validation has been discussed in detail by Breiman *et al.*[5] We have also studied the convergence empirically with a simulated dataset in Sec. 5.1.

4.6. *Problem of multiple local minima: Use of simulated annealing*

IFS may suffer from similar problems like neural networks because of the presence of possibly numerous local minima in the transformed feature space. Therefore, it may be worthwhile to start with different starting points or to use simulated annealing. We opted for the second one to keep the algorithm automatic. During feature extraction, each time we compute two competitive direction vectors, one as discussed in Sec. 4.2 and another by simulation from a uniform distribution (over a unit hyper-sphere with the current value of $\boldsymbol{\alpha}_i$ as origin). After normalization, we chose the one which has the smaller training set RSS as the new estimate $\boldsymbol{\alpha}_i^*$. If this new estimate reduces the current value of RSS, Z_i is modified. Otherwise, we draw a random number u from $U(0,1)$ and adjust Z_i only when $u \leq \exp\left(-\frac{\text{RSS}^* - \text{RSS}}{\text{RSS} \times C_m}\right)$, where RSS^* is the residual sum of squares corresponding to $\boldsymbol{\alpha}_i^*$. This step helps to overcome the problem of getting stuck at bad local minima. C_m is a constant that depends on the number of iteration (m). Generally, a large number is chosen as C_0 to make almost all the directions acceptable in the beginning. It is then decreased gradually over iterations to attain convergence. Several options are available for determining these constants (see Ref. 28), but we use a simple one. We start with $C_0 = \frac{100}{p}$, where p is the number of measurement variables, and then use $C_m = U(0.8, 0.99) \times C_{m-1}$ $(m = 1, 2, \ldots)$ to modify it. During modification of direction vectors $\boldsymbol{\alpha}$ and features ϕ, at any stage, we always store the linear combinations $Z_1, Z_2, \ldots, Z_p$ for which training set RSS is minimum up to that stage.

5. Experimental Results

In this section, we use some simulated and benchmark data sets to illustrate the usefulness of the proposed method. In Tables 1 and 2, we report the misclassification rates of IFS algorithm starting with two different sets of direction vectors as mentioned in Sec. 4.4 and denote them by IFS (variables) and IFS (features), respectively. For assessing the accuracy of IFS, we compare their performances with a number of existing parametric and nonparametric classifiers. Results of linear discriminant analysis (LDA), quadratic discriminant analysis (QDA), CUS, method of successive projections, neural networks (with sigmoidal transformation function), radial basis function networks (RBF), kernel discriminant analysis, nearest neighbor classifiers and support vector machines (SVM) are reported to facilitate the comparison. Error rates for classification tree method are also reported in benchmark data sets. For simulated data sets, we compute the optimal error rates and present them in Table 1. These optimal error rates represent the misclassification rate of the optimal Bayes rule applied to specific data sets.

For kernel discriminant analysis (see e.g. Ref. 22), we standardized the observations of a class by using the usual moment based estimate of class dispersion matrix and then used same bandwidth in all directions. Optimal bandwidth for a class density estimate was selected by least square cross-validation (see e.g. Refs. 38 and 40). To find density estimates at a new data point, at first we standardized it using the same standardization matrices used earlier to get the density estimate at the standardized data points. Density estimates at the original data point can be computed from that using a simple formula when the measurement vector undergoes a linear transformation. For nearest neighbor classification (see e.g. Refs. 10 and 18), we used Mahalanobis distance (see e.g. Ref. 30) as the distance function, which is equivalent to use Euclidean distance after standardization by estimated pooled dispersion matrix.

For the neural networks (single hidden layer perceptrons with sigmoidal transformation) and the radial basis function (RBF) networks, we standardized the measurement vectors before using training algorithms. For the perceptrons, we tried both backpropagation (see e.g. Ref. 35) and Levenberg–Marquardt algorithms (see e.g. Ref. 39) and opted for the latter because of its better performance both in terms of computing time and misclassification rate. On each data set, we trained the model for several choices of hidden nodes, and reported the best error rate that we got in the test set. Similarly, we used various choices of bandwidth in the case of radial basis function networks, and several choices of bandwidth and cost parameter in the case of support vector machines, and reported their best performances on test sets. Matlab programs available in Matlab neural network toolbox and Matlab SVM toolbox were used for running these three algorithms. In the case of SVM, for multiclass problems, voting was used to take the final decision after all pairwise comparisons. We used the S-Plus package for building the classification trees. Codes for other classification algorithms were written in C-language.

5.1. *Results from the analysis on simulated data sets*

Here, we consider three simulated examples on two-class classification problems. The first two of these data sets contain two-dimensional measurement vectors, while in Example 3, we consider a higher dimensional problem. Since training sets for real problems are usually not very large (because of the cost involving generation of samples), we use relatively smaller training sets in our simulations. However, to obtain reliable estimates for misclassification rates, much larger test sets are used. For each of these examples, we generate 500 observations for training and 3000 for the test sets taking equal number of observations from the two classes. Each experiment is carried out ten times, and average misclassification rates of different classifiers over these ten simulation runs are reported in Table 1 along with their corresponding standard errors.

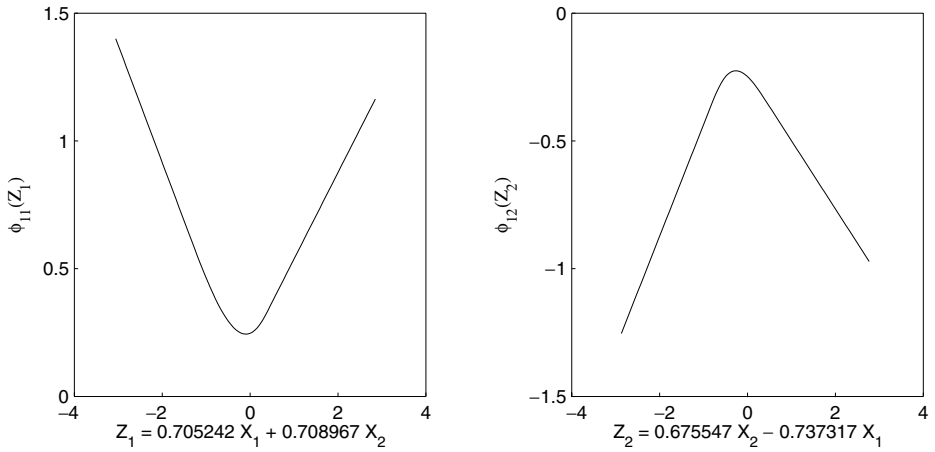
Example 1. In this example, each class is an equal mixture of two bivariate normal distributions differing only in their location parameters. The population density functions are given by

$$\begin{aligned} p(\mathbf{x}|1) &= 1/2[N_2(1,1,0.6^2,0.6^2,0) + N_2(-1,-1,0.6^2,0.6^2,0)] \\ p(\mathbf{x}|2) &= 1/2[N_2(1,-1,0.6^2,0.6^2,0) + N_2(-1,1,0.6^2,0.6^2,0)] \end{aligned} \tag{12}$$

Clearly, $X_1X_2 = 0$ is the optimal class boundary for this problem. Therefore, the ideal linear combinations (linear features) are of the form $Z_1 = \alpha_1X_1 + \alpha_2X_2$ and $Z_2 = \alpha_1X_1 - \alpha_2X_2$ ($\alpha_1^2 + \alpha_2^2 = 1$), whereas the corresponding features (transformation functions) are quadratic in nature. It is evident from Table 1 that IFS performed quite well in this example, and in fact, it could estimate the ideal linear combinations and transformation functions very accurately in all ten simulation runs, one of which is chosen below for illustration.

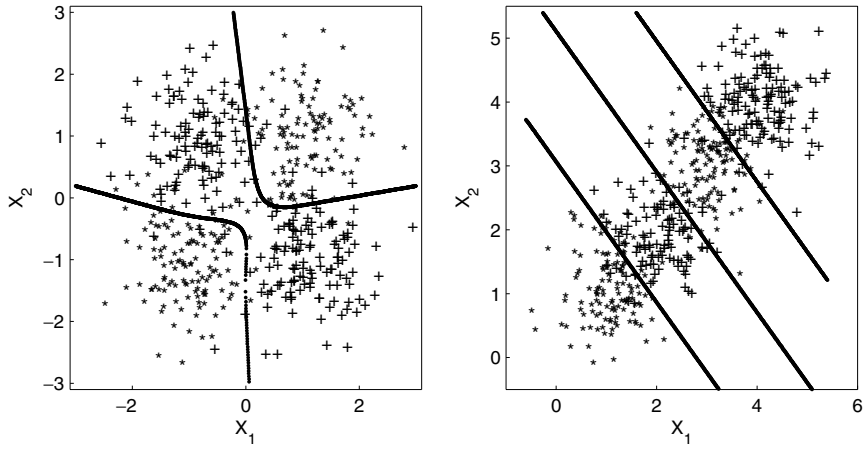
In this example, linear combinations Z_1 and Z_2 were estimated as $0.705X_1 + 0.709X_2$ and $0.676X_2 - 0.737X_1$. Due to additivity constraint, as expected, we got $\widehat{\phi}_{1i} = -\widehat{\phi}_{2i}$ (for $i = 1, 2$) and $\widehat{\phi}_{10} + \widehat{\phi}_{20} = 1$. Moreover, the desired quadratic nature of the plotted functions $\widehat{\phi}_{11}$ and $\widehat{\phi}_{12}$ (see Fig. 1) clearly suggests the success of IFS in estimating the ideal functions as well. Though instead of actual values of X_1X_2, we used only the indicator variables, IFS was still able to detect the proper linear combinations and the corresponding nonlinear features (transformation functions). Hence the estimated boundaries in Fig. 2(a) also turned out to be quite satisfactory.

Table 1 shows that in this example, performance of LDA was not satisfactory at all but error rates for QDA came closest to the optimal error rates. Classification methods like CUS and successive projections, which are based on additive regression did not work at all. However, IFS and other nonparametric classifiers could nearly match the performance of QDA. As the optimal class boundary ($X_1X_2 = 0$) is not an additive function of the measurement variables, in a few cases IFS got stuck to bad local minima when original variables are used as the initial features. Simulated annealing helped in such situations. In this example, successive projections could not improve upon CUS, but IFS did a remarkable job for this purpose.



(a) Feature 1 (b) Feature 2

Fig. 1. Estimated features in Example 1.



(a) Example 1 (b) Example 2

Fig. 2. Estimated class boundaries in Examples 1 and 2.

Example 2. Like the previous example, here also competing classes are equal mixtures of bivariate normal populations but overlaps between the competing classes occur in a different manner. Population density functions of these two populations are given by

$$\begin{aligned} p(\mathbf{x}|1) &= 1/2[N_2(1,1,0.5^2,0.5^2,0) + N_2(3,3,0.5^2,0.5^2,0)] \\ p(\mathbf{x}|2) &= 1/2[N_2(2,2,0.5^2,0.5^2,0) + N_2(4,4,0.5^2,0.5^2,0)] \end{aligned} \tag{13}$$

Since the true class boundary is highly nonlinear in this example, neither LDA nor QDA could manage to estimate it properly (see Table 1). CUS and the method of

Table 1. Average misclassification rates and their standard errors (in percentage) in simulated examples.

	Example-1	Example-2	Example-3
Optimal	9.04 (0.15)	12.08 (0.16)	10.74 (0.14)
LDA	49.18 (0.50)	45.23 (0.57)	49.98 (0.33)
QDA	9.24 (0.11)	45.44 (0.54)	11.61 (0.17)
CUS	48.23 (0.83)	18.35 (0.41)	14.30 (0.27)
Succ. Proj.	44.55 (1.83)	18.54 (0.40)	14.02 (0.28)
Nearest neighbor	9.62 (0.13)	13.22 (0.19)	16.49 (0.20)
Kernel	9.80 (0.14)	12.86 (0.23)	12.43 (0.16)
SV-Machine	9.67 (0.14)	12.41 (0.18)	13.04 (0.18)
Neural network	9.74 (0.15)	12.58 (0.21)	12.48 (0.19)
RBF network	9.46 (0.16)	12.33 (0.17)	12.92 (0.17)
IFS (variables)	9.76 (0.17)	12.58 (0.18)	12.31 (0.18)
IFS (features)	9.66 (0.20)	12.38 (0.18)	11.88 (0.21)

successive projections showed much better performance, yet the misclassification rates were far from optimum. However, other nonparametric classifiers achieved reasonably low misclassification rates. Among these classifiers, nearest neighbor method had slightly higher error rate but performances of other classifiers were fairly similar. It is quite evident from Fig. 2(b) that IFS could identify the optimal class boundary. Though successive projections failed to improve the performance of CUS, IFS was very successful in this problem again.

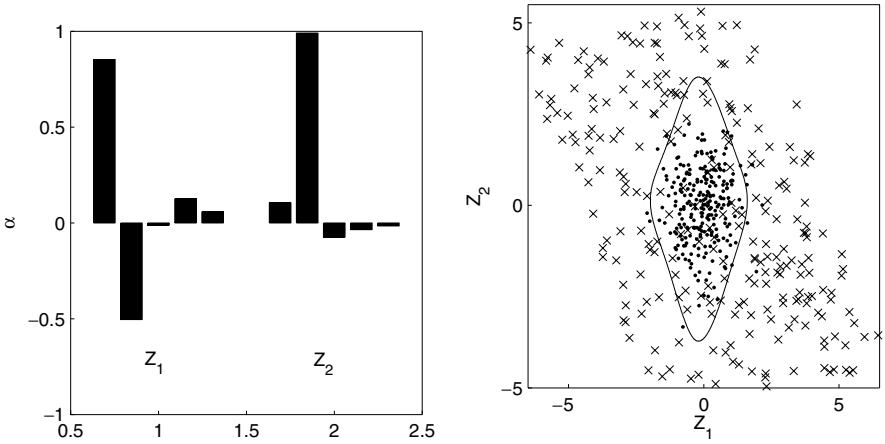
Example 3. Next we consider a higher dimensional classification problem between two populations, where the first two co-ordinate variables in the two classes are distributed as

$$p(\mathbf{x}|1) = N_2(0, 0, 1, 1, 1/2) \quad p(\mathbf{x}|2) = U[-5, 5] \times U[-5, 5] \tag{14}$$

Other three variables are independent and identically distributed as $N(0, 1)$ in both the populations. These variables do not contain any separability information, and they are used simply to add noise.

Clearly, in this example, the true class boundary is of the form $X_1^2 + X_2^2 - X_1X_2 = C_0$ for some constant C_0. This can be rewritten (not uniquely) as $A\{(X_1 - 0.5X_2)\}^2 + BX_2^2 = C$, where A, B and C are appropriate constants. Thus, we see that only two linear features contain the information about class separability. When we ran IFS algorithm on different sets of 500 observations generated from these two-classes, the final model contained exactly two linear features in most of the cases. Figure 3 shows that IFS could obtain very appropriate estimates of linear features and that of the optimal class boundary in this noisy example. Due to quadratic nature of the true class boundary, as expected, QDA led to the best performance in this example. IFS performed better than all other parametric and nonparametric classifiers and the corresponding error rates were close to that of QDA.

Since the optimal error rate is known for a simulated problem, we chose to perform an empirical study. In Example 1, we trained IFS on training sets of varying



(a) Estimated features (b) Estimated class boundary

Fig. 3. Estimated features and class boundaries in Example 3.

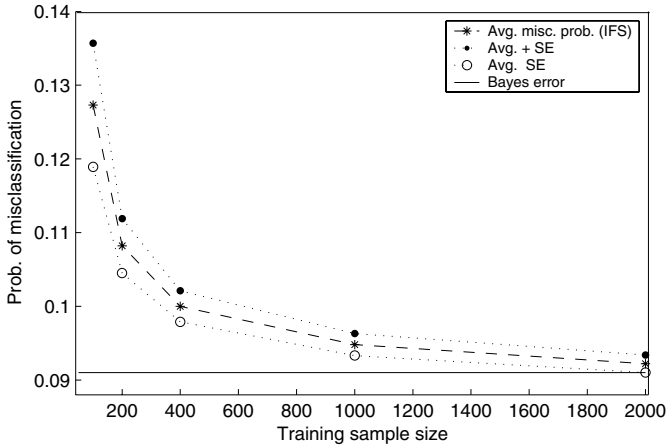


Fig. 4. Misclassification rates of IFS for different sample sizes.

sizes consisting of 100, 200, 400, 1000 and 2000 cases, and computed the test set errors based on fixed size of 3000 test cases. Figure 4 shows the average misclassification rate over these ten simulation runs and the corresponding standard error for each training sample size. This figure clearly indicates that the test set error rate of IFS converges quite quickly to some value very close to the optimal error rate.

5.2. *Results from the analysis on benchmark data sets*

We analyze five benchmark data sets for further illustration of the proposed method. All these data sets has previously been used by many other authors (see e.g. Refs. 3, 9, 23, 24, 34 and 36), who evaluated the performance of many classifiers on these

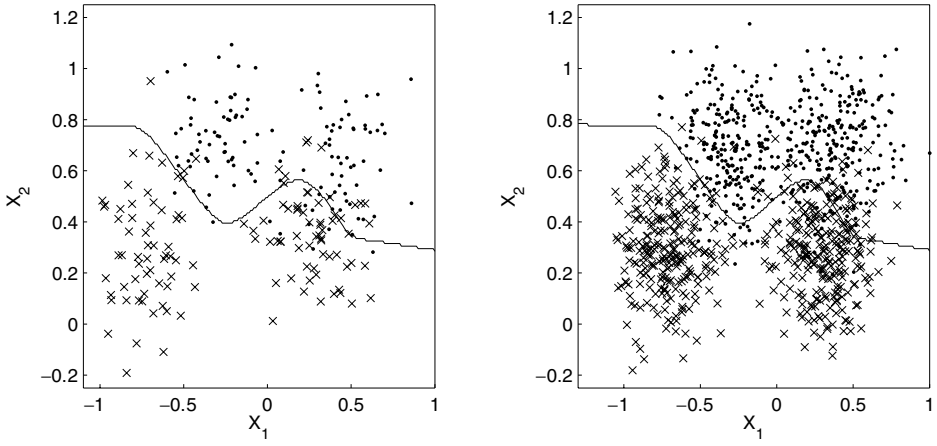
data sets. Along with the error rates of all parametric and nonparametric classifiers that we considered in Sec. 5.1, here we report the misclassification rates for classification tree methods as well. Throughout this section, sample proportions of different classes are used as their prior probabilities. The data sets that we use in this section have specific training and test samples. The sizes of the training and test sets for each data set have been reported in Table 2, which also shows the test set error rates for different classification methods. A brief description of these data sets is given below.

Synthetic data. It consists of bivariate observations from two competing classes, each of which is an equal mixture of two bivariate normal populations. All these normal populations have the same dispersion matrix but different location parameters. These parameters were chosen to have a Bayes risk of 8.0%. A scatter plot of this data set is given in Fig. 4. This data set is available at CMU data archive (http://lib.stat.cmu.edu).

We have used two data sets related to vowel recognition problem and refer them as vowel data-1 and vowel data-2.

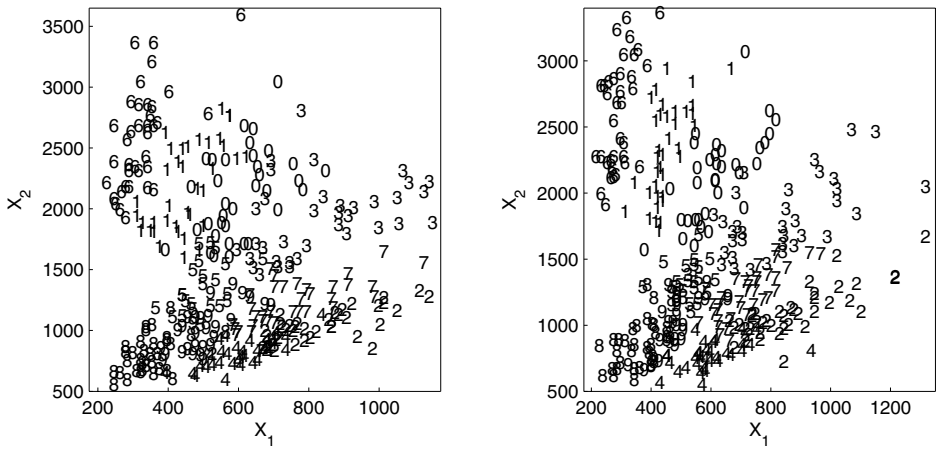
Vowel data-1. This data was created by Peterson and Barney[33] by a spectrographic analysis of vowels in words formed by "h" followed by a vowel and then followed by "d". There were 67 persons who spoke different words and the two lowest resonant frequencies of a speaker's vocal track were noted for ten different vowels. A scatter plot of this data set is given in Fig. 6, where the numbers represent the labels of different classes ("0" represents the tenth class).

Image data. It originally contains 19 measurements on each image of one of seven different objects : brickface, sky, foliage, cement, window, path and grass. This



(a) Training set: 250 observations (b) Test set: 1000 observations

Fig. 5. Estimated class boundaries for synthetic data.



(a) Training set: 338 observations (b) Test set: 333 observations

Fig. 6. Scatter plots for vowel data-1.

data set and the description of the variables are available at UCI machine learning repository (http://www.ics.uci.edu/∼mlearn). The value of the variable 'region pixel count' is '9' for all observations. For other two variables, "short line density-5" and "short line density-2", almost 95% of the values are zero. We did not consider these three variables in our study. There are some variables in the data set which are linear or nonlinear functions of three other variables: R ("raw red mean"), B ("raw blue mean") and G ("raw green mean"). We have deleted those variables too and carried out our analysis using the remaining nine variables (region-centroid column, region-centroid-row, vedge-mean, vedge-sd, hedge-mean, hedge-sd, rawread-mean, rawblue-mean and rawgreen-mean).

Satimage data. The sample database was generated by taking a small section from the original Landsat Multi-Spectral Scanner Image Data purchased from NASA by the Australian Centre for Remote Sensing. The database consists of the multispectral values of pixels (coded as numbers) in 3×3 neighborhoods in a satellite image, and the classification associated with the central pixel in each neighborhood. However, for our analysis we used only four spectral values related to central pixel to classify it into one of six different classes : red soil, cotton crop, grey soil, damp grey soil, soil with vegetation stubble and very damp grey soil. The data set and its description is available at UCI machine learning repository.

Vowel data-2. This data set is also available at UCI machine learning repository. Like vowel data-1, it is also related to a vowel recognition problem, where ten measurements are taken for each observation coming from any of 11 classes. The description of the measurement variables can be found in Ref. 23. There are 528 observations in the training set and 462 observations in the test set, which are equally distributed among 11 competing classes.

Table 2. Misclassification rates (in percentage) for benchmark data sets.

	Synthetic	Vowel-1	Satimage	Image	Vowel-2
Dimension	2	2	4	9	10
No. of classes	2	10	6	7	11
Train. sample size	250	338	4435	210	528
Test sample size	1000	333	2000	2100	462
LDA	10.80	25.23	19.30	11.14	55.63
QDA	10.20	19.82	15.65	14.62	52.81
CUS	8.50	24.32	19.55	10.10	51.30
Succ. Proj.	8.50	18.92	16.90	7.48	45.67
Nearest Neighbor	11.70	17.72	15.35	8.24	46.75
Classification Tree	10.10	23.72	19.50	12.57	56.28
Kernel	9.30	18.32	15.30	15.71	62.12
SV-Machine	9.40	21.02	14.80	11.95	46.54
Neural network	9.40	18.62	16.95	9.95	48.92
RBF network	9.40	20.42	15.70	14.00	49.78
IFS (variables)	9.10	20.72	15.80	6.76	44.37
IFS (features)	9.60	23.42	15.40	6.28	43.29

Table 2 reveals that IFS performed quite well in all these data sets. On synthetic data, CUS and successive projection methods had the best error rates but IFS and all other nonparametric methods except classification tree and nearest neighbor classifier worked well. Class boundary estimated by IFS (features) is shown in Fig. 5. On vowel data-1, nearest neighbor method led to the best error rate. Kernel discriminant analysis and successive projection methods also had reasonably lower error rate. Performances of IFS and other nonparametric methods were fairly satisfactory. On satellite image data (satimage data), SVM yielded the lowest misclassification rate but the error rates for QDA, kernel discriminant analysis, nearest neighbor, RBF neural network and IFS were very close to that. Performance of IFS was much better than CUS and successive projection methods. On image segmentation data (image data), IFS performed extremely well, and the test set misclassification rates for the two different starting points were much lower compared to other parametric and nonparametric classification techniques. Given that the test sample size was quite large, the difference between the error rate of IFS and that of the other classifiers was found to be statistically significant. On vowel data-2, once again IFS led to best error rates among the parametric and nonparametric classifiers considered here. Successive projections could reduce the misclassification rate of CUS in this example quite a bit, but IFS did even better. This data set was used extensively by Hastie and Tibshirani and Hastie *et al.*,[23] where they reported results of many other classification methods. IFS performed better than every other method except for FDA-MARS (deg-2) which had a slightly lower error rate. Overall the error rate of IFS was either the lowest among the methods compared or was very close to the lowest error rate for a given problem. IFS also had the lowest average test set error across all the examples.

Since different classification methods were run in different platforms in our experiments, it was not possible to compare their CPU times. Perhaps with the

Table 3. Average number of iterations and CPU times taken by IFS algorithms for training the network on different data sets (figures in top and bottom rows are for IFS(variables) and IFS(features), respectively). These averages are taken over 11 runs including ten times for cross-validation.

	Example-1	Example-2	Example-3	Synthetic	Vowel-1	Satimage	Image	Vowel-2
No. of iterations	15.9	12.8	22.4	11.7	22.9	24.7	94.2	83.5
	13.5	12.7	33.7	9.3	10.5	30.9	91.5	67.6
CPU time (secs.)	0.74	0.63	2.92	0.28	2.05	86.05	28.53	133.80
	0.65	0.62	4.33	0.23	1.07	108.87	28.38	104.39

evolution of faster computers, the training time is not as important as it used to be. However, IFS (implemented in C-programming language) works reasonably fast. Since it is an iterative algorithm, it is difficult to find its computational complexity. If we consider the dimension d and the number of class J to be finite (which is usually the case) for each iteration IFS requires $O(K^2n)$ calculations, where K is the number of knots placed on the range of each variable and n is the training sample size. The number of knots should increase with the sample size but at an extremely slow rate. Hence, the computational cost per iteration is only marginally higher than $O(n)$. Table 3 below shows the CPU time (on a Pentium-4 machine) and average number of iterations required by IFS algorithm on different data sets. For instance, on vowel data-2 it took around 20 minutes to train the model 11 times including ten times for cross-validation. However, on the same platform, for a single run of neural network (a two-layer perceptron with 20 hidden nodes), MATLAB took almost half an hour (for 100 iterations) for training. For complex problems, generally a large number of hidden nodes is required and hence it takes a long time to train the network. But IFS deals with fewer number of linear features which helps to reduce the computing cost substantially.

6. Conclusion

In real life classification problems, the class boundaries are more complex rather than being linear or quadratic. In such cases, traditional methods of discriminant analysis often turn out to be inadequate. This article presents a nonparametric method (IFS) for discriminant analysis, which is capable to approximate these class boundaries using a model similar to the projection pursuit regression model. It automatically tries to identify some linear combination of the original measurement variables and fits an additive model based on regression splines using those combinations. Model selection (selection of the ideal number of linear combinations and the number of knots for fitting splines) is done using cross-validation.

IFS can be viewed as a generalized version of the single hidden layer perceptron model, which uses sigmoidal or other known transformation functions at the hidden layer. On the other hand, instead of using any fixed transformation function, IFS uses cubic splines to estimate the transformation functions from the data itself which makes it more flexible.

The use of backfitting for convergence makes the algorithm fast. This enables use of cross-validation for model selection which requires a number of training on different partitions of the data. Model selection by backward deletion allows for very simple models.

IFS can also be viewed as a modification of CUS (see Ref. 3), which uses an additive model based on the original measurement variables. The experiments reported in Sec. 5 clearly show the effectiveness of IFS over CUS in terms of lower misclassification rates in various classification problems. Another advantage of being a generalization of the CUS procedure is that IFS does not have to use random starting points. If one starts with $\boldsymbol{\alpha}_i^0$ which are unit vectors along co-ordinate axes (i.e. original variables as initial linear features), the initial result will be the result of CUS which usually yields reasonable misclassification rates. The subsequent iterations can only improve the results further. Thus one can expect convergence to a better solution.

The experiments also show that IFS has a definite edge over perceptron models both in terms of misclassification rate and also in visualizing the class separability. It could find out the ideal direction vectors and nonlinear features in all examples that we have tried including the ones reported here. IFS also fared well compared to many other nonparametric claasification techniques. Finally the entire procedure is fully automatic. Only the initial number of knots has to be specified.

IFS can be adopted for regression problems also. Instead of indicator variables, the response variables can be used to estimate the regression surface by projection pursuit regression models. One of the major issues in projection pursuit regression is to find the proper linear combinations. IFS uses backfitting to estimate them iteratively and automatically. A similar work using smoothing splines is available in Ref. 36, which perhaps could be improved upon using a model selection procedure such as the one used in IFS.

Acknowledgment

We are thankful to an associate editor and four anonymous referees who carefully read an earlier version of the paper and provided us with several helpful comments.

Appendix

IFS Algorithm:-

||**Initialization**||
Define indicator variables $Y_1, Y_2, \ldots, Y_J$ for J competing classes.
Initialize the direction vectors $\boldsymbol{\alpha}_1^0, \boldsymbol{\alpha}_2^0, \ldots, \boldsymbol{\alpha}_p^0$ $(||\boldsymbol{\alpha}_i^0|| = 1 \; \forall \; i \;)$ and compute the linear combinations $Z_i^0 \leftarrow \boldsymbol{\alpha}_i^{0'} \mathbf{x}, \quad i = 1, 2, \ldots, p.$
For each Z_i^0, place the knots $t_{i_1}^0, t_{i_2}^0, \ldots, t_{i_K}^0$ on its range to define the basis functions

$$B_{00}^0 \leftarrow 1, \; B_{i0}^0 \leftarrow Z_{i_0}^0, \; B_{ik}^0 \leftarrow (Z_i^0 - t_{i_k}^0)_+^3, \quad i = 1, 2, \ldots, p; \; k = 1, 2, \ldots, K.$$

Regress Y_j $(j = 1, 2, \ldots, J)$ on the basis functions to estimate ϕ_{j0}^0 , $\phi_{j1}^0, \ldots, \phi_{jp}^0$.

Compute probability estimates $\widehat{Y}_{j_n}^0 \leftarrow \phi_{j0}^0 + \sum_{i=1}^{p} \phi_{ji}^0(Z_{i_n}^0)$ and residuals
$r_{j_n}^0 \leftarrow Y_{j_n} - \widehat{Y}_{j_n}^0, \ \ (n = 1, 2, \ldots, N)$.
Count $\leftarrow 0$; Stop $\leftarrow 0$; $m \leftarrow 0$, $\text{RSS}_0 \leftarrow \sum_{j=1}^{J} \sum_{n=1}^{N} (r_{j_n}^0)^2$.
while(Stop=0) do

||**Backfitting**||
$\text{RSS}_a \leftarrow \text{RSS}_0$
for $(i = 1$ to $p)$ do
Use Taylor series approximation (up to linear term) of RSS about $\boldsymbol{\alpha}_i^m$

$$\text{RSS} \simeq \sum_{j=1}^{J} \sum_{n=1}^{N} [r_{j_n}^m - \boldsymbol{\delta}_i^{'} \xi_{j_n}]^2,$$

where $\boldsymbol{\delta}_i = \boldsymbol{\alpha}_i - \boldsymbol{\alpha}_i^m$ and $\xi_{j_n} = \frac{\partial \phi_{ji}(\boldsymbol{\alpha}_i^{'} \mathbf{x})}{\partial \boldsymbol{\alpha}_i} | \boldsymbol{\alpha}_i = \boldsymbol{\alpha}_i^m, \mathbf{x} = \mathbf{x}_n$.
Use least squares method to estimate $\boldsymbol{\delta}_i$.
Compute $\boldsymbol{\alpha}_i^{m*} \leftarrow \frac{\boldsymbol{\alpha}_i^m + \boldsymbol{\delta}_i}{||\boldsymbol{\alpha}_i^m + \boldsymbol{\delta}_i||}$ and $Z_i^{m*} \leftarrow \sigma(\boldsymbol{\alpha}_i^{m*'} \mathbf{x})$.
Place the knots $t_{i_k}^{m*}$ on Z_i^{m*} to compute basis functions $B_{ik}^{m*}, \ (k = 0, 1, \ldots, K)$.
Regress $Y_j - \sum_{k<i} \phi_{jk}^{m+1}(Z_k^{m+1}) - \sum_{k>i} \phi_{jk}^m(Z_k^m)$ on these basis functions to estimate ϕ_{j0}^{m*} and $\phi_{ji}^{m^*}, \ (j = 1, 2, \ldots, J)$.
Compute $\widehat{Y}_{j_n}^{m*}$, $r_{j_n}^{m*}, \ (j = 1, 2, \ldots, J; \ n = 1, 2, \ldots, N)$ and RSS^* accordingly.
if($\text{RSS}^* \leq \text{RSS}_a$)
$\text{RSS}_a \leftarrow \text{RSS}^*, \ \ \boldsymbol{\alpha}_i^{m+1} \leftarrow \boldsymbol{\alpha}_i^{m*}; \ Z_i^{m+1} \leftarrow Z_i^{m*}$.
$B_{ik}^{m+1} \leftarrow B_{ik}^{m*}, \ \ (k = 1, 2, \ldots, K)$.
$\phi_{ji}^{m+1} \leftarrow \phi_{ji}^{m*}; \ \phi_{j0}^m \leftarrow \phi_{j0}^{m*}, \ (j = 1, 2, \ldots, J)$.
$\widehat{Y}_{j_n}^m \leftarrow \widehat{Y}_{j_n}^{m*}; \ r_{j_n}^m \leftarrow r_{j_n}^{m*}, \ (j = 1, 2, \ldots, J; \ n = 1, 2, \ldots, N)$.
end(if)
if($\text{RSS}^* > \text{RSS}_a$)
$\boldsymbol{\alpha}_i^{m+1} \leftarrow \boldsymbol{\alpha}_i^m; \ Z_i^{m+1} \leftarrow Z_i^m; \ \phi_{ji}^{m+1} \leftarrow \phi_{ji}^m, \ \ (j = 1., 2, \ldots, J)$.
end(if)
end(for)
$\phi_{j0}^{m+1} \leftarrow \phi_{j0}^m; \ \widehat{Y}_{j_n}^{m+1} \leftarrow \widehat{Y}_{j_n}^m; \ r_{j_n}^{m+1} \leftarrow r_{j_n}^m, \ (j = 1, 2, \ldots, J; \ n = 1, 2, \ldots, N)$.
$\text{RSS}_1 \leftarrow \text{RSS}_a$; Reduction $\leftarrow \frac{\text{RSS}_1 - \text{RSS}_0}{\text{RSS}_0}$.
if(Reduction ≤ 0.001)
Regress $Y_j \ \ (j = 1, 2, \ldots, J)$ on the current basis functions to re-compute $\phi_{j0}^{m+1}, \phi_{j1}^{m+1} \ldots, \phi_{jp}^{m+1}$.
Re-adjust $\widehat{Y}_{j_n}^{m+1}, r_{j_n}^{m+1}, \ (n = 1, 2, \ldots, N)$ and RSS_1.
Reduction $\leftarrow \frac{\text{RSS}_1 - \text{RSS}_0}{\text{RSS}_0}$.
end(if)
$m \leftarrow m + 1$; $\text{RSS}_0 \leftarrow \text{RSS}_1$.
||**Termination**||
if(Reduction > 0.001) Count $\leftarrow 0$
if(Reduction ≤ 0.001) Count $\leftarrow$ Count + 1
if((Reduction=0) or (Count=Count_stop)) Stop $\leftarrow 1$
end(while)

- Count_Stop is a user defined parameter for stopping criterion. In this article, we use Count_Stop = 5 and terminate the program if relative reduction in RSS is insignificant (i.e. Reduction < 0.001) over five consecutive iterations. However, the final result is not very sensitive to this parameter if any larger value is used.
- For the sake of simplicity, the simulated annealing part has not been included in the above description, but one can easily incorporate it following the idea described in Sec. 4.6.

References

1. T. W. Anderson, *An Introduction to Multivariate Statistical Analysis* (Wiley, New York, 1984).
2. S. Bose, A method for estimating nonlinear class boundaries in the classification problem and comparison with other existing methods, PhD Dissertation. (Dept. of Stat., Univ. of California, Berkeley, 1992).
3. S. Bose, Classification using splines, *Comput. Stat. Data Anal.* **22** (1996) 505–525.
4. S. Bose, Multilayer statistical classifier, *Comput. Stat. Data Anal.* **42** (2003) 685–701.
5. L. Breiman, J. H. Friedman, R. A. Olshen and C. J. Stone, *Classification and Regression Trees* (Wadsworth & Brooks, Monterrey, California, 1984).
6. L.Breiman and J. H. Friedman, Estimating optimal transformations for multiple regression and correlation (with discussion), *J. Amer. Stat. Assoc.* **80** (1985) 580–619.
7. L. Breiman, Fitting additive models to regression data: diagnostics and alternating views. *Comput. Stat. Data Anal.* **15** (1993) 13–46.
8. B. Cheng and D. M. Titterington, Neural networks: a review from a statistical perspective (with discussion), *Stat. Sci.* **9** (1994) 2–54.
9. C. A. Cooley and S. N. MacEachern, Classification via kernel product estimators, *Biometrika* **85** (1998) 823–833.
10. T. M. Cover and P. E. Hart, Nearest neighbour pattern classification, *IEEE Trans. Inform. Th.* **13** (1968) 21–27.
11. N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines* (Cambridge University Press, Cambridge, 2000).
12. C. de Boor, *A Practical Guide to Splines* (Springer-Verlag, New York, 1978).
13. P. A. Devijver and J. Kittler, *Pattern Recognition: a Statistical Approach* (Prentice Hall, London, 1982).
14. R. Duda, P. Hart and D. G. Stork, *Pattern Classification* (Wiley, New York, 2000).
15. R. A. Fisher, The use of multiple measurements in taxonomic problems. *Ann. Eugen.* **7** (1936) 179–188.
16. J. Friedman and W. Stuetzle, Projection pursuit regression, *J. Amer. Stat. Assoc.* **76** (1981) 817–823.
17. J. H. Friedman, On bias, variance, 0-1 loss, and the curse of dimensionality, *Data Min. Knowl. Discov.* **1** (1997) 55–77.
18. E. Fix and J. L. Hodges, Jr., Discriminatory analysis, nonparametric discrimination, consistency properties. Randolph Field, Texas, Project 21-49-004, Report No. 4, 1951.
19. K. Fukunaga, *Introduction to Statistical Pattern Recognition* (Academic Press, New York, 1990).
20. A. K. Ghosh and S. Bose, Backfitting neural networks, *Comput. Stat.* **19** (2004) 193–210.

21. A. K. Ghosh and P. Chaudhuri, On data depth and distribution free discriminant analysis using separating surfaces, *Bernoulli* **11** (2005) 1–27.
22. D. J. Hand, *Kernel Discriminant Analysis* (Wiley, Chichester, 1982).
23. T. Hastie, R. Tibshirani and A. Buja, Flexible discriminant analysis, *J. Amer. Stat. Assoc.* **89** (1994) 1255–1270.
24. T. Hastie, R. Tibshirani and J. H. Friedman, *The Elements of Statistical Learning*: *Data Mining, Inference and Prediction* (Springer Verlag, New York, 2001).
25. K. Hornik, M. Stinchcombe and H. White, Muti-layer feed forward networks are universal approximators, *Neural Networks* **2** (1989) 359–366.
26. P. J. Huber, Projection pursuit (with discussion), *Ann. Stat.* **13** (1985) 435–525.
27. C. Kooperberg, S. Bose and C. J. Stone, Polychotomus regression, *J. Amer. Stat. Assoc.* **92** (1997) 117–127.
28. P. J. M. Laarhoven and E. H. L. Aarts, *Simulated Annealing*: *Theory and Application* (D. Reidel Pub, Dordrecht, 1987).
29. D. J. C. Mackay, A Bayesian framework for backpropagation networks, *Neural Computation* **4** (1992) 448–472.
30. P. C. Mahalanobis, On the generalized distance in statistics, *Proc. Nat. Acad. Sci. India* **12** (1936) 49–55.
31. G. J. McLachlan, *Discriminant Analysis and Statistical Pattern Recognition* (Wiley, New York, 1992).
32. S. Mika, G. Ratch, J. Weston, B. Scholkopf, A. J. Smola and K.-R. Muller, *Invariant feature Extraction and Classification in Kernel Spaces*, Advances in Neural Information Processing System, Vol. 1, eds. A. J. Smola, T. K. Leen and K.-R. Muller (MIT Press, 2000), pp. 526–532.
33. G. E. Peterson and H. L. Barney, Control methods used in a study of vowels, *J. Acoust. Soc. Amer.* **24** (1952) 175–185.
34. B. D. Ripley, Neural networks and related methods for classification (with discussion), *J. Roy. Stat. Soc. Series B* **56** (1994) 409–456.
35. B. D. Ripley, *Pattern Recognition and Neural Networks* (Cambridge University Press, Cambridge, 1996).
36. C. B. Roosen and T. Hastie, Automatic smoothing spline projection pursuit, *J. Comput. Graph. Stat.* **3** (1994) 235–248.
37. S. Scholkopf, C. J. C. Burges and A. J. Smola, *Advances in Kernel Methods: Support Vector Learning* (MIT Press, Cambridge, 1999).
38. D. W. Scott, *Multivariate Density Estimation*: *Theory, Practice and Visualization* (Wiley, New York, 1992).
39. G. A. F. Seber and C. J. Wild, *Nonlinear Regression* (Wiley, New York, 1989).
40. B. W. Silverman, *Density Estimation for Statistics and Data Analysis* (Chapman and Hall, London, 1986).
41. M. Stone, Cross validation: A review, *Mathematische Operationsforschung und Statistik, Series Statistics* **9** (1977) 127–139.
42. V. Vapnik, *Statistical Learning Theory* (Wiley, New York, 1998).
43. M. Zhu, Feature extraction and dimension reduction with applications to classification and the analysis of co-occurrence data. Ph.D. Dissertation, Stanford University, (2001).

Anil Kumar Ghosh received the B.Sc (Hons) degree in statistics from the University of Calcutta in 1996. He received the MStat and PhD degrees from the Indian Statistical Institute, Kolkata, in 1998 and 2004, respectively.

During 2004–2006, he was a postdoctoral research fellow at the Institute of Statistical Science, Academia Sinica, Taiwan and a research associate at the Mathematical Sciences Institute, Australian National University, Canberra. Currently, he is an Assistant Professor at the Mathematics and Statistics Department in Indian Institute of Technology, Kanpur.

The fields of his research interests include pattern recognition, nonparametric and robust statistics, machine learning and statistical computing.

Smarajit Bose received the B. Stat and M. Stat degrees from the Indian Statistical Institute, Calcutta in 1984 and 1986 respectively, and the Ph. D. degree from the University of California, Berkeley in 1992, all in Statistics.

Between 1991 and 1992, he was a visiting scientist in IBM Almaden Research Center, San Jose where he worked with the Advance Process Monitoring Group. He visited the Statistics Departments of the University of Washington, Seattle during 1992–93 and the Ohio State University, Columbus during 1993-96 where he worked on classification and clustering problems and their applications in Medical Imaging and Ergonomics. In 1996, he joined the Indian Statistical Institute, Calcutta and now is a Professor in the Applied Statistics Division of the Institute. He has also visited the University of California, Santa Barbara in 2001, and during 2002-03.

His current research interests are Pattern Recognition and its applications in Image Processing and Speaker Identification.