A Feature-Based Serial Approach to Classifier Combination

Mark Last[1]

Department of Information Systems Engineering, Ben-Gurion University of the Negev, Beer-Sheva 84105, ISRAEL

mlast@bgumail.bgu.ac.il

Horst Bunke

Institut fur Informatik and angewandte Mathematik, University of Bern, Neubrückstrasse 10, CH-3012, Bern, Switzerland

bunke@iam.unibe.ch

Abraham Kandel

Department of Computer Science and Engineering, University of South Florida, 4202 E. Fowler Avenue, ENB 118, Tampa, FL 33620 USA

kandel@csee.usf.edu

[1] Corresponding author

A Feature-Based Serial Approach to Classifier Combination

Abstract

A new approach to serial multi-stage combination of classifiers is proposed. Each classifier in the sequence uses a smaller subset of features than the subsequent classifier. The classification provided by a classifier is rejected only if its decision is below a pre-defined confidence level. The approach is tested on a two-stage combination of *k*-Nearest Neighbor classifiers. The features to be used by the first classifier in the combination are selected by two stand-alone algorithms (Relief and Info-Fuzzy Network, or IFN) and a hybrid method, called "IFN + Relief." The feature-based approach is shown empirically to provide a substantial decrease in the computational complexity, while maintaining the accuracy level of a single-stage classifier or even improving it.

Keywords: classifier combination, sequential combination, feature selection, nearest neighbor classifier, decision-tree classifier, Info-Fuzzy Network (IFN)

Originality and Contribution

The main motivation behind existing methods of classifier combination is improvement of classification accuracy. In this paper we are concerned with a different problem, which is improvement of *computational efficiency*. Reducing computational effort is extremely important in data-intensive applications of pattern recognition such as clickstream analysis on the web. The classification rate of "lazy" learning methods, like *k-NN* Classifier, is especially sensitive to the dimensionality of the training set. The computational complexity of a *k-NN* Classifier depends on two main factors: the number of distance computations (proportional to the number of training examples considered for finding the nearest neighbor of a new instance) and the effort associated with calculating a distance to each training example (proportional to the number of features).

Several algorithms have been suggested to reduce the number of distance computations. To the best of our knowledge, this work presents the first attempt to reduce the computational complexity

of each distance calculation by decreasing the *number of features* used in the process. Our approach is motivated by the observation that in many applications the majority of classified instances are 'well-behaved'. This means they can be classified using a relatively small portion of available features, while just for a few 'hard' cases a more sophisticated classifier requiring more features is needed. Consequently, we are introducing here a novel, *feature-based* approach to serial combination of two *k-NN* classifiers. To the best of our knowledge, no serial combination of k-NN classifiers (feature-based or other) has even been proposed.

The features sufficient for classifying "well-behaved" instances can be identified by automated methods of feature selection such as IFN (Info-Fuzzy Network) and Relief. We also present here a novel feature selection method, termed "IFN + Relief." In this method, the *number* of the selected features is determined by IFN, while the *selection* of features is based on the relevance level calculated by Relief.

To sum-up, this work is original in at least the following aspects: it uses a feature-based approach to reducing the computational burden of an instance-based classifier; it proposes a serial combination of two *k-NN* classifiers; and it presents a new feature selection method ("IFN + Relief"). Evaluation results demonstrate the broad practical implications of the proposed methodology.

1 Introduction

Classifier combination has become a very active area of research [21]. The driving force behind these activities is the expectation that classification errors can be corrected if an ensemble of classifiers rather than a single classifier is used for a given task. It has been proved, in fact, that for the case of independent classifiers the error rate of the overall system monotonically decreases as more classifiers are added to a system, provided each individual classifier has an error rate that is better than random guessing [13] [30]. Hence, in theory, a combination of weak classifiers can achieve any desired degree of classification accuracy.

A number of different architectures for classifier combination have been proposed. The one that is most frequently used is a parallel combination. Here a number of classifiers $C_1,...,C_n$ individually make a decision about the class of an unknown input pattern. All these decisions are fed into a

combiner. The combination rules employed by the combiner include functions such as product, sum, mean, median, maximum and minimum of a posteriori probabilities and related quantities [20]. More sophisticated approaches use trainable classifiers as combiners [7]. If each classifier does not return any *a posteriori* probabilities, but just a single class, or the *n* best ranked classes, combination strategies based on voting or Borda count are commonly used [22].

Rather than combining the outputs of multiple classifiers, the *Dynamic Classifier Selection (DCS)* approach (see [11][12]) suggests to select the best single classifier for each test pattern. A common objective function for the dynamic selection of a classifier is the so-called *Classifier Local Accuracy (CLA)*, which can be estimated in a local region of the feature space [11]. The underlying assumption of the DCS approach is that given a test pattern, there is at least one classifier that can classify it correctly.

Another architecture that can be employed for classifier combination is hierarchical concatenation. This approach is particularly useful if many pattern classes are to be distinguished (e.g., in the recognition of Chinese characters). Here classification is achieved by routing an unknown pattern through a chain of classifiers where each classifier reduces the number of feasible classes. Decision tree classifiers are a special case of this architecture. They correspond to the case where each classifier in the hierarchical chain uses only a single feature of the input pattern or a pre-defined function of selected features (see [4] [14] [28]).

Serial combination is an alternative strategy for combining decisions of multiple classifiers [29]. The principal feature of this strategy is that the individual classifiers are applied sequentially. Hence, at each stage (layer) there is only one classifier classifying the patterns. There are two basic approaches to the serial combination of classifiers: *class set reduction approach* and *reevaluation approach*. Under the first approach, the number of possible classes is reduced continuously, while the second approach requires reevaluation of the patterns, which are rejected at the preceding stage. The decision of a classifier in a serial combination is rejected if its confidence level falls below a pre-defined threshold. Usually, the initial layers represent coarser decisions than the final layer. As indicated by [29], the overall classification performance of such a system can exceed the performance of any of its individual layers only if classifiers appearing at the different layers use different feature spaces and different discriminant functions.

The main motivation behind parallel and serial combination architectures is improvement of classification accuracy. In this paper we are concerned with a different issue, namely, improvement of computational efficiency through classifier combination. We use a serial classifier combination scheme that follows the reevaluation approach (see [29] and Chapter 11 of [31]). Our system is motivated by the observation that in many applications the majority of the patterns to be classified are 'well-behaved'. This means they can be classified using a relatively small portion of available features, while just for a few 'hard' cases a more sophisticated classifier requiring more features is needed. But if the sophisticated classifier is applied to all patterns, including the easy ones, more resources than necessary are spent. The feature-based approach to classifier combination is based on a common observation that the classification accuracy of most classifiers is a non-decreasing function of the number of input features (see [24]).

The architecture of our system is shown in Figure 1. There is a sequential chain of classifiers C_1,..., C_n of growing complexity, i.e., classifier C_1 is the simplest and C_n the most sophisticated in the ensemble. First, C_1 is activated on an input pattern. Whenever classifier C_{i-1} rejects the input, the next classifier in the chain, C_i, is called, $2 \leq i \leq n$.

The architecture shown in Figure 1 allows many different implementations. In the approach proposed in this paper we consider only two stages, i.e., two different classifiers, C_1 and C_2. The first classifier uses just a subset of the available features. Whenever the decision reached by C_1 is below a given level of confidence, C_1 rejects the input and activates C_2. For the purpose of simplicity, we assume that C_2 is a classifier of the same type as C_1, but uses the full set of available features. For most classifiers the computation time grows with the dimensionality of the feature space. Therefore, C_1 will be faster than C_2. If a sufficiently large portion of all patterns will be classified by C_1, and not passed onto C_2, substantial savings in computation time can be expected. Of course, variation of the confidence threshold in C_1 allows trading computation time for classification accuracy.

The architecture shown in Fig. 1 requires for each classifier C_i to select a subset of features F_i out of all available features F, such that $F_1 \subseteq F_i \subseteq F_n = F$. Many different approaches to feature selection have been proposed in the literature. For an overview see, for example, Chapter 8 of [33]. There are optimal and sub-optimal procedures for feature selection. Optimal methods are typically based on search procedures with heuristic pruning strategies. These procedures are prone to combinatorial

explosion and restricted to low-dimensional problems. Sub-optimal methods trade optimality for computational efficiency. The two basic strategies are top-down (also called sequential backward elimination) and bottom-up (also known as sequential forward selection). Under the first strategy, one starts with the full set of features and sequentially eliminates the weakest feature, while bottom-up methods first chooses the single best feature and iteratively add the best out of the remaining features. More advanced methods consider sets rather than single features at a time and combine forward and backward search with each other [27] [32].

A number of linear dimension reducers have been developed over years. The linear methods of dimensionality reduction include *projection pursuit* [10], *factor analysis* [18], and *principal components analysis* [8]. These methods are not aimed directly at eliminating irrelevant and redundant variables, but are rather concerned with transforming the observed variables into a small number of "projections", or "dimensions". Thus, the linear methods cannot reduce directly the number of features used by a classifier as long as all the variables have non-zero weights in the linear combination, though the features with low weights are often ignored.

In order to measure the quality of features, two basic procedures have been proposed in the literature. The first, which is known as feature filter [16], evaluates a feature, or a set of features, based on some performance criterion. Examples of such performance criteria are divergence, Chernoff, Battacharrya, Patrick-Fisher, or Mahalanobis distance, which measure the separability of class distributions (see Chapter 8 in [33]). The second procedure, called the wrapper model in [16], directly uses a classifier to measure the quality of a set of features. In this paper, we limit our discussion to the filter approach, which usually requires less computational effort than the wrapper model [24], though a wrapper model based on a serial classifier system itself may be considered in the future.

Kira and Rendell [19] have suggested a feature filter algorithm, called Relief, which evaluates each feature by its ability to distinguish among instances that are near each other. Their selection criterion, the *feature relevance*, is applicable to numeric and nominal features. The threshold of relevancy is determined statistically by using Chebyshev's inequality, which may be not sharp enough for making a clear distinction between relevant and non-relevant features. Another shortcoming of the Relief algorithm is its inability to identify redundant features within the set of

relevant features. Consequently, the set of features selected by Relief may be non-optimal for a given classifier. Nevertheless, Relief was used in this work as one of the feature selection methods.

In [23], we have introduced a novel, information-theoretic method of feature selection, based on the information-theoretic fuzzy approach to knowledge discovery [25]. The method integrates feature selection with a highly scalable data mining algorithm, leading to elimination of both irrelevant and redundant features. The relevant features are clearly identified by including only a subset of available features in the resulting model, called *Info-Fuzzy Network (IFN)*. The efficiency of the information-theoretic method for feature selection has been demonstrated in [23] on a single-stage decision-tree classifier (C4.5). In this paper, the method is integrated, for the first time, with a *k*-Nearest Neighbors Classifier and multiple classifier combination.

The rest of our paper is organized as follows. In Section 2, we describe the feature selection algorithms used in this work. A two-stage serial combination of *k*-Nearest Neighbor Classifiers is presented in Section 3. Empirical evaluation of the system is carried out in Section 4 and a comparative study with similar approaches aimed at increasing classifier efficiency is presented in Section 5. Conclusions regarding possible extensions of our approach are drawn in Section 6.

2 Feature Selection Algorithms

As indicated in Section 1 above, the first classifier in our two-stage combination schema uses a subset of available features selected by a feature selection algorithm. The necessary properties of such an algorithm include applicability to both discrete and continuous features and computational efficiency. As indicated above, only the filter model algorithms were considered. Out of a large number of possible choices, we have adopted two existing algorithms, Relief [19] and IFN [23][25] and developed a new hybrid algorithm called "IFN + Relief". The three algorithms are described in the next sub-sections.

2.1 The Relief Algorithm

The Relief algorithm, proposed by Kira and Rendell in [19], selects features by their relevance to the target concept (function). The basic idea of Relief is similar to the guiding principle of the *k*-Nearest Neighbors algorithm: instances that are closer to a given instance are more likely to belong to the same class. If a dimension (feature) is relevant, the closest instances of the same class are

expected to be closer to a given instance *along that dimension* than the closest instances of all the other classes. Consequently, the *relevance level* of a given feature *i* is calculated in [19] by:

$$\left(\sum_{j=1}^{m} -diff(x_{ij}, near\_hit_{ij}) + diff(x_{ij}, near\_miss_{ij}) \right) \Big/ m \qquad (1)$$

where *m* is the sample size (a randomly selected subset of the training set) and *diff* (x_{ij}, $near\_hit_{ij}$) is the difference between the value of the feature *i* in a randomly picked instance *j* and the value of the feature *i* in the nearest training example having the same class ($near\_hit_{ij}$). Correspondingly, $near\_miss_{ij}$ is defined as the value of *i* in the nearest training example having a different class. For relevant features, the values of x_{ij} and $near\_hit_{ij}$ are expected to be very close, while the values of x_{ij} and $near\_miss_{ij}$ are expected to be different. If a feature is irrelevant, both differences are expected to have nearly the same distribution.

Since Relief normalizes the values of *diff* to the [0, 1] range, the average relevance level of a relevant feature should be close to one, but it cannot be higher than one. On the other hand, the relevance levels of irrelevant features should be close to zero, though, as indicated by [19], they tend to take slightly negative values. The threshold τ used to select or discard a given feature depends on the required significance level α and the sample size *m*. The maximum value of τ is computed as follows (based on Chebyshev's inequality):

$$\tau = \frac{1}{\sqrt{\alpha\, m}} \qquad (2)$$

According to [19], the distinction between relevant and irrelevant features can also be determined by manual inspection, which requires running Relief in a "semi-automatic" mode. We solve this problem by a novel method called "IFN + Relief" (see sub-section 2.3 below).

If the sample size (the number of training instances used for the calculation of the relevance level) is kept constant, the run time of Relief is linear in the number of features and the total number of training instances [19]. In a general case, one would like to use the entire training set for calculating the relevance levels, which makes the complexity of Relief quadratic in the number of instances. The complexity is increased because finding the nearest hit and the nearest miss requires calculating the distance between every two instances in the training set.

2.2 The Info-Fuzzy Network (IFN) Algorithm

An Info-Fuzzy Network (IFN) [25] is a classification model, which has a root node, a changeable number of hidden layers (one layer for each selected feature), and a target layer representing the possible classes. Each hidden layer consists of nodes representing different values of an input feature. The network differs from the structure of a standard decision tree (see [4][28]) in two aspects: it is restricted to the same feature at all nodes of the same hidden layer and it has interconnections between the terminal (leaf) nodes and the final nodes, which represent the classes under consideration. The connectionist nature of the model resembles the structure of a multi-layer neural network (see [26]). Consequently, we define this model as a *network* and not as a *tree*.

An example of a two-layered network (based on two selected features) is shown in Figure 2. The first selected feature has three values, represented by nodes 1,2, and 3 in the first layer, but the network construction algorithm split only nodes 1 and 3. The second layer has four nodes standing for the combinations of two values of the second selected feature with two split nodes of the first layer. The three classes to be distinguished are represented by three nodes in the target layer. Also, there exist 15 connections, denoted by dashed lines, between the five terminal (unsplit) nodes and the three target nodes.

The network construction procedure starts with a single-node network representing an empty set of selected features. In the original version of the information-theoretic algorithm [25], a node in the network is split only if it provides a statistically significant decrease in the *conditional entropy* [5], which measures the uncertainty of the target class Y, given the values of the features $X_1, \ldots, X_n$. In this paper, we are using a less restrictive requirement that the conditional entropy should be greater than zero, since the statistical significance testing proved to be less favorable for the performance of the selected features in the *k*-Nearest Neighbor Classifier.

A decrease in the conditional entropy of a random variable is termed *conditional mutual information* [5]. The conditional mutual information ($MI(A_{i'}; A_i / z)$) of an original feature *i'* and a target attribute *i*, given a node *z* is estimated by the following expression (based on [5]):

$$MI(A_{i'}; A_i / z) = \sum_{j=0}^{M_i - 1} \sum_{j'=0}^{M_{i'} - 1} P(V_{ij}; V_{i'j'}; z) \bullet \log \frac{P(V_{i'j'}^{ij} / z)}{P(V_{i'j'} / z) \bullet P(V_{ij} / z)} \quad (3)$$

where

- M_i and $M_{i'}$ denote the number of values of the target attribute *i* and the original feature *i'* respectively. The algorithm discretizes continuous features to a finite number of intervals by a dynamic discretization procedure, which is based on [9].
- $P(V_{i'j'}/z)$ is the estimated conditional (*a posteriori*) probability of a value *j'* of the original feature *i'*, given the node *z*.
- $P(V_{ij}/z)$ is the estimated conditional (*a posteriori*) probability of a value j of the target attribute *i*, given the node *z*.
- $P(V_{i'j'}^{ij}/z)$ is the estimated conditional (*a posteriori*) probability of a value *j'* of the original feature *i'* and a value *j* of the target attribute *i*, given the node *z*.
- $P(V_{ij}; V_{i'j'}; z)$ is the estimated joint probability of a value *j* of the target attribute *i* , a value *j'* of the original feature *i'*, and the node *z*.

The network is built by a greedy stepwise procedure: at each step, a new feature is selected to maximize the total decrease in the conditional entropy, as a result of splitting the nodes of the last layer. The nodes of a new hidden layer are defined for a Cartesian product of split nodes of the previous hidden layer and the values of the new selected feature. If there is no feature decreasing the conditional entropy of the target attribute, the network construction stops.

The information-theoretic method of feature selection is highly scalable. It is shown in [25] that, for discretely valued features, the run time of the network construction procedure is linear in the number of training examples and quadratic in the number of original features. Moreover, it is reduced by a factor of $p(2-p)$, where p is the proportion of features selected by the algorithm out of the total number of features. For continuous features, the algorithm uses a dynamic discretization procedure, which is quadratic-logarithmic in the number of training examples. Thus, in large discrete datasets, IFN is expected to run faster than Relief, which is quadratic in the number of examples and linear in the number of features. However, with an increase in the number of continuous features, Relief should become the faster algorithm. The actual run times of the algorithms on datasets of variable size and feature type are compared in sub-section 4.4 below.

2.3 IFN + Relief

As mentioned in sub-section 2.1 above, the Relief algorithm does not provide a clear threshold for distinguishing between relevant and irrelevant features. Consequently, Relief cannot be used in a fully automatic mode. On the other hand, the output of the IFN algorithm specifies a "crisp" number of selected features. Thus we apply here a novel feature selection method, which we call "IFN + Relief." According to the new method, the *number* of the selected features is determined by IFN, while the *selection* of features is based on the relevance level calculated by Relief. As shown empirically in Section 4 below, the performance of IFN + Relief is significantly better than the performance of the stand-alone Relief method. It is also comparable with the performance of the stand-alone IFN.

3 Two-Stage Combination of *k*-Nearest Neighbor Classifiers

In this section, we summarize the properties of a single-stage *k*-Nearest Neighbor (*k*-NN) Classifier and describe the implementation and the main settings of the two-stage classifier, based on the *k*-NN algorithm. We also suggest a new measure for the computational complexity of a multi-stage classification process.

3.1 *K*-Nearest Neighbor Algorithm

The *k*-Nearest Neighbor (*k-NN*) Classifier is a "lazy" learning algorithm: rather than estimating the target function for the entire instance space, it estimates it locally for each new instance to be classified [26]. Thus, the *k*-NN Classifier is characterized by a negligible training cost vs. a relatively high cost of classifying every new instance. Once all the training examples have been stored in the computer memory, the process of instance classification includes the following three steps:

1) Calculate the distance between the new instance and every example in the training set by using a standard Euclidean measure or any other distance measure available.
2) Find the *k* training examples, which are the nearest neighbors of the instance to be classified. The number of nearest neighbors *k* is a user-defined parameter.

3) Classify the instance by the most common class in the set of k training examples. If k is equal to one, the predicted class of a new example will be identical to the class of its nearest neighbor.

The reasoning behind the algorithm is straightforward: since most target functions are likely to change gradually rather than abruptly in the feature space, the nearest (most similar) examples should have the same label as the new instance. The main disadvantage of the k-NN approach is the high computational cost of computing the distance between the input and the training patterns, based on the values of *all* available features. Moreover, if the target concept depends on only few of the many available features, the distance between similar instances may be quite large, causing a decrease in the classification performance of the algorithm [26].

3.2 Two-Stage Classifier

In this paper, we propose a *two-stage* k-NN Classifier, which is implemented as follows (see Figure 1 below). The input to the system includes the training set and the subset of features selected by a feature selection algorithm. The first classifier C_1 attempts to classify a new instance by using the selected features only. The number of the nearest neighbors in the majority class is then compared to a user-defined threshold, which represents a minimum percentage of the total number of nearest neighbors (k). If the percentage of examples in the majority class is greater than the threshold, the system outputs the classification made by C_1. Otherwise (if the majority is less than or equal to the threshold), the second, more expensive, classifier C_2 is applied by using the full set of the available features. In the second case, the system outputs the classification made by C_2.

The system implementation involves the choice of the following parameters:

- *Number of nearest neighbors (k).* This parameter has to be chosen for any *k-NN* algorithm. While k can significantly affect the classification performance for a given dataset, its impact on the computational complexity of the k-NN Classifier is relatively small, since the bottleneck of the algorithm is calculating the distances between the input and all training examples rather than choosing a subset of the nearest examples. To simplify the choice of other parameters for our experiments (see below), we have set the number of nearest neighbors to the constant value of

10. This number seems large enough to study the effect of the majority threshold on the system performance.

- *Performance measure.* The traditional criterion of maximum classification accuracy would usually cause us to prefer a single-stage classifier, disregarding its computational cost. However, we are interested to reduce the computational complexity of the classification process without a significant decrease in the resulting accuracy. A new measure of computational complexity, called "complexity ratio", is suggested in the next sub-section. The trade-off between accuracy and complexity is analyzed in Section 4 by using Receiver-Operating Curves (ROC).

- *Feature selection algorithm*. We need an algorithm, which can identify the most relevant features that provide nearly the same classification performance as the full set of features. The number of selected features should be minimal, since smaller subsets of selected features improve the efficiency of the first classifier (C_1). In addition to the two existing feature selection algorithms (Relief and IFN), we have introduced here a novel combination of these algorithms called "IFN + Relief." All three algorithms are described in Section 2 above.

- *Majority threshold.* This is the highest percentage of the training instances belonging to the most common class, which makes C_1 to reject its result. If the value of the threshold is less than *1/m*, where *m* is the number of classes, C_2 will never be activated, since the majority percentage cannot be less than *1/m*. On the other hand, the threshold of *1.0* will always cause the result of C_1 to be rejected. As the threshold gets higher, we expect the run time of the system to increase, since more and more instances are classified by the second classifier (C_2). At the same time, the classification performance of the system is expected to improve and to get closer to the accuracy of C_2.

3.3 Evaluating Computational Complexity

The computational complexity of a multi-stage classifier on a given dataset can be evaluated directly by its actual run time. However, the run time is not a reliable performance measure, since

it depends on several factors, including the computational power of a specific CPU, the size of computer memory, and the amount of available resources on a given computer. In the case of an *n*-stage *k*-NN Classifier, a more objective measure of the required computational effort is the total number of feature-values, *TFV*, accessed for classifying *X* instances, which is given by the following formula:

$$TFV_n = \sum_{i=1}^{n} m_i x_i \qquad (4)$$

where n is the total number of classifiers, m_i is the number of features (out of M) used by classifier i ($\forall i: 0 \le m_i \le m_{i+1} \le M$), and x_i is the number of instances classified by classifier i ($\Sigma x_i = X$) .

The number of feature-values accessed by a single-stage *k*-NN Classifier is $TFV_1 = M*X$. Consequently, the number of feature-values used by different classifiers can be normalized by calculating the *Complexity Ratio* R_c as follows:

$$R_c = TFV_n \,/\, TFV_1 = \frac{\sum_{i=1}^{n} m_i x_i}{M * X} \qquad (5)$$

4 Experimental Evaluation of the System

To evaluate the performance of the proposed system, we have applied the two-stage *k*-NN Classifier to seven benchmark datasets, available from the UCI Machine Learning Repository [3]. According to [3], all these datasets are based on real-world data. The datasets chosen for the system evaluation vary both in feature type (purely discrete, purely continuous and mixed) and in dimensionality of the data. The datasets include between eight and 1,024 available features, while the number of instances varies from 178 to 3,196. The maximum number of classes is ten. Approximately two thirds of each set have been chosen randomly as a training set, while the remaining instances were held out for testing the classifier. The characteristics of the datasets are summarized in Table 1.

4.1 Overview

The experiments with the system included the following issues:

- *Classification performance of selected features.* Only the training instances of each dataset were used in the feature selection process. The three feature selection methods (Relief, IFN, and IFN + Relief) were evaluated by comparing the classification accuracy of C_1 on the testing sets of instances. At this stage, no majority threshold was applied, to eliminate the activation of the second classifier (C_2) for any instance. The accuracy of the full feature set classifier C_2 on the same testing set was used as a reference.
- *Classification accuracy vs. computational complexity.* A given set of selected features can be used by the two-stage classifier with different values of the majority threshold, which varies between *1/m*, where *m* is the number of classes and 1.0. In each dataset, we used two types of charts to represent graphically the trade-off between accuracy and complexity. The first chart was the Receiver-Operating Curve (ROC), which showed complexity vs. accuracy for the sets of features selected by each method. The ROC charts were used to identify the best feature selection method for each dataset. For the best method, we built another chart showing complexity and accuracy as a function of the majority threshold.
- *Computation times.* The ultimate goal of the two-stage classifier is to reduce classification time without a significant loss of the classification accuracy. Thus, we conclude our work with studying the actual run times of the two-stage classifier as a function of the complexity ratio. We also compare the training times of the feature selection algorithms (Relief and IFN). All runs were performed on the same computer with approximately the same amount of available resources. As indicated in sub-section 3.3 above, the run time alone cannot be considered an objective performance measure due to various reasons. However, it provides an important indication regarding the overall contribution of the proposed approach to the efficiency of the classification process.

The detailed results of the experiments are presented and analyzed in the next sub-sections.

4.2 Classification Performance of Selected Features

In Table 2 below, we show the classification accuracy of single-stage *k*-NN Classifiers, which use the following sets of features:

- A full set of features (equivalent to the second classifier C_2 in the two-stage combination)
- Features selected by Relief
- Features selected by IFN
- Features selected by IFN + Relief

The second column in the table shows the relevancy threshold τ applied to the output of the Relief algorithm. The value of τ was determined by manual inspection of bar charts showing the relevance levels of all the features sorted in descending order of relevance. An example of such bar chart, obtained for the Wine dataset, is shown in Figure 3 below. In the next two columns of Table 2, we show the number of features selected by using Relief and IFN as stand-alone methods. As explained above, the number of features used by the hybrid IFN + Relief method is equal to the number of features selected by IFN. The number of selected feature is also presented as the percentage of the total number of available features. We proceed with presenting the testing error of a single-stage *k*-NN Classifier using all the available features. The testing error rates for the three methods of feature selection are shown in the next three columns. As previously indicated, all classifiers were run with the number of nearest neighbors $k = 10$. An asterisk is shown next to an error rate, if it is significantly higher than the error rate with the full set of features (at the 95% confidence level).

If we compare the "stand-alone" feature selection methods (Relief and IFN), IFN seems to produce a more accurate classifier than Relief: the average error rate of IFN-based classifier (0.242) is lower than the average error rate as a result of applying Relief (0.316). However, with Relief, the error rate is significantly higher than with the full set of features in four datasets only vs. five datasets with IFN. The performance of the hybrid approach (IFN + Relief) is slightly worse than IFN (the average error rate is 0.279), though in one dataset (Chess), the classification accuracy with the hybrid method is even higher than with the full set of features. Like in the case of IFN, the

classifier based on the hybrid approach was significantly worse than the full set classifier in five datasets out of seven.

To sum-up, none of the feature selection methods under consideration proves to be consistently better or consistently worse than the other two. Using a single-stage classifier with a selected set of features, we can save, on average, between 79% and 89% of the computational effort related to the number of features, at the cost of a significant decrease in the classification accuracy vs. a classifier based on the full set of features. In the next sub-section, we show how the two-stage classifier combination can maintain or even improve the accuracy without using a full set of features for classifying most instances.

4.3 Accuracy vs. Complexity

As indicated in sub-section 3.2 above, majority threshold is the highest percentage of the neighboring training instances in the majority (most common) class, which is needed to reject the output of the first classifier C_1. As long as the majority class is not larger than this threshold, an instance is classified by the second (full feature set) classifier C_2. In a given dataset, the threshold may vary from the inverse of the number of classes to one. With an increase in the threshold value, more and more instances are classified by C_2. Consequently, the complexity ratio R_C goes up, accompanied by a possible decrease in the testing error rate E. Both complexity ratio and the corresponding error rate are affected by the set of selected features used by C_1. A set of features can be considered superior to another set of features, if it provides a lower error rate with the same complexity or, alternatively, a lower complexity with the same error rate for *any* majority threshold. To evaluate the feature selection methods included in our experiment, we have built the Receiver-Operating Curves (ROC), which show the error rate as a function of complexity ratio. For the best sets of features, we have also presented the error rate and the complexity ratio as a function of the majority threshold. A summary of the obtained results is provided below.

Chess Dataset. The ROC chart of the results is given in Figure 4. The "IFN+Relief" method for feature selection shows here a clear advantage over the stand-alone IFN and Relief methods. Moreover, the two-stage system proves to be *more accurate* than the single-stage classifier, *along with* having a lower computational complexity. According to Figure 5, for any majority threshold of 0.9 and lower, the error rate is smaller than the error of a single-stage classifier, while the complexity ratio does not exceed 40%.

Credit Dataset. According to the ROC chart in Figure 6, Relief and IFN+Relief are the best feature selection methods for this dataset. Although in this case, the two-stage system cannot reach the accuracy of the full feature set classifier, the trade-off between accuracy and complexity is clearly demonstrated in Figure 7 for the features selected by the Relief algorithm. Particularly, we can get close to the accuracy of the single-stage classifier by less than 0.5% while using only about 40% of its complexity.

Diabetes Dataset. Here the Relief algorithm seems to provide the best performance, though the distinction between the curves is not so clear (see Figure 8). Figure 9 shows an important result that with the majority threshold of 0.7 we can decrease the error rate below the single-stage level, while higher thresholds move it back to that level, but not higher. In either case, the complexity ratio may be kept below 70%.

Glass Dataset. Figure 10 clearly shows that the features of this dataset selected by IFN provide a significantly better performance than the features selected by the two other methods. As shown by Figure 11, the majority threshold of 0.6 causes the error rate to drop slightly below the single-stage level. If the threshold is increased beyond 0.6, the error rate goes up again, but still stays in the same range

Heart Dataset. Here, IFN+Relief has a clear advantage (see Figure 12). Similar to the Chess Dataset, the accuracy of the two-stage system surpasses the accuracy of the single-stage classifier for several values of the majority threshold (see Figure 13). At the same time, the complexity ratio can be kept below 50%.

Opt_Orig Dataset. As shown by Figure 14, IFN is the best feature selector for this dataset. Though the two-stage system cannot improve the accuracy of the single-stage classifier, we can get within 0.2% of its accuracy by using less than 50% of the single-stage computational complexity. The trade-off between accuracy and complexity is demonstrated in Figure 15.

Wine Dataset. According to Figure 16, Relief and IFN are very close to each other in this dataset, though the behavior of the former method seems less consistent. The trade-off between the complexity and the error rate of IFN is shown in Figure 17. The accuracy of the single-stage classifier can be reached with the complexity ratio of nearly 50% which corresponds to the majority threshold of 0.9

The main conclusion from the results, presented in this sub-section, is that the proposed approach to classifier combination provides an efficient way of trading recognition accuracy for computational complexity. As shown above, the complexity can be reduced by 50% and more without sacrificing any significant recognition accuracy. Moreover, in some datasets, the two-stage classifier combination even improved the recognition rate of a single-stage classifier, *along with* a lower complexity. Unfortunately, no feature selection algorithm has been found consistently better or consistently worse than the other two, which means that each dataset requires some "fine-tuning" in terms of finding the best set of features by one of the algorithms and the best value of the majority threshold. In the next sub-section, we study the effect of the reduced complexity on the actual classification time of the system.

4.4 Computation Times

The computation times of the stand-alone feature selection algorithms used (Relief and IFN) are shown in the two rightmost columns of Table 3 below. The hybrid method (IFN + Relief) does not require re-running of either algorithm. Hence, its run time can be calculated as the sum of times of both methods. Table 3 also presents again the size of the training set, used by the algorithms and the number of candidate (original) features of each type (discrete / continuous). All the run times were obtained on a Pentium III computer with a 500 MHz processor and 128 MB RAM.

The results presented in Table 3 confirm the theoretical analysis of the computational complexity made in Section 2. In purely discrete datasets (Chess and Optdigits-Orig), IFN, which is only linear in the number of training instances, runs much faster than Relief, which is quadratic in the number of instances. The difference goes up with an increase in the size of the training set. However, the presence of continuous features slows down the IFN algorithm with respect to Relief, which is linear in the number of features of *any* type. The examples include Credit, Diabetes, Glass, Heart, and Wine datasets.

Figure 18 and Figure 19 below show the time ratio of a two-stage classifier as a function of the complexity ratio. The time ratio is calculated as the ratio between classification times of a two-stage classifier and a single-stage classifier (C_2), which uses a full set of features. The classification time of a single-stage classifier was 363.06 seconds for the Opt_Orig Dataset and 14.00 seconds for the Chess Dataset. The first classifier (C_1) was based on the features selected by the "IFN + Relief" method. All the times refer to classifying the "testing examples" (about 1/3 of

each dataset). The classification times cannot be directly combined with the training times of the feature selection algorithms (see Table 3 below), since in a real-world pattern recognition task (like character or digit recognition), the dataset to be classified is expected to be *much* larger than the set of training examples. As demonstrated by Figure 18 and 19 below, there is a strong linear relationship ($R^2 \approx 0.999$) between the complexity ratio of a two-stage system and its actual classification time.

4.5 Discussion of Results

The underlying assumption of the feature-based approach to serial classifier combination is that using a full set of features is likely to improve the classification accuracy vs. using only a subset of relevant features. The definition of relevancy depends on the feature selection algorithm in use. In sub-section 4.2 above, we have tested our assumption on three different feature selection algorithms (Relief, IFN, and IFN+Relief) and seven benchmark datasets. On average, any one of the algorithms under consideration has lead to a decrease in the classification accuracy. Relief was the worst one: the features selected by Relief have doubled the average error rate. In contrast, IFN has raised the average error rate by 57% only (from 0.155 to 0.242). It is also worth noting that Relief has selected fewer features than IFN in most datasets.

The main purpose of our system is trading accuracy for complexity. The system can only be useful if a considerable reduction in its complexity can be reached at an "affordable" cost in terms of classification accuracy. The ROC curves in sub-section 4.3 above show us that in most datasets, a minimal error rate, which is either close to or even below the single-stage error, can be achieved with the complexity ratios of 70%, 60%, and even 40%. In other words, our method can save up to 60% of the computational effort without sacrificing any accuracy at all! A downside of our results is that obtaining the best trade-off between accuracy and complexity requires both selecting the best feature selection algorithm for a given dataset and finding the optimal value of the majority threshold. On the other hand, any efforts for improving the efficiency of a k-NN classifier (like selecting an optimal subset of prototypes in [34]) can only be justified if the classifier is about to classify a sufficiently large number of patterns. Thus, optimizing the design parameters of our system before applying it to a large set or stream of patterns may well be carried out on a representative subset of the data in question.

The computational complexity of the system has been evaluated in sub-section 4.3 by a theoretical parameter called "total number of feature-values" (*TFV*), which we have defined in sub-section 3.3. However, the actual measure of computational complexity of a k-NN classifier is the total classification time. In sub-section 4.4 above, we have shown that our theoretical measure of computational complexity is strongly correlated with the actual classification time of the system. Thus, *TFV* can be reliably used as a criterion for optimizing the overall performance of a multi-stage serial classifier.

5 Comparative Study

Reducing the amount of computation associated with classifying new instances is very important for "lazy" learning methods, like k-nearest neighbors, which defer the processing of training examples until each new instance becomes available. Thus, the efficiency of the k-Nearest Neighbor classifier has been a subject of many works in pattern recognition literature. To the best of our knowledge, no serial combination of k-NN classifiers (feature-based or other) has ever been proposed. However, as we show below, some other useful ideas for decreasing the computational burden have been discussed in literature.

An efficient technique for reducing the number of distance computations in a k-NN classifier is proposed in [1]. According to this approach, the data points are clustered into several subsets of equal size. In the classification stage, all subsets that are too far from a test pattern are eliminated from the search for the nearest neighbor. A similar approach of partitioning the feature space into cells is presented in [6], where the distance computations are reduced to calculating the distance between a test sample and the center of gravity of each cell. Unlike the algorithms of [1] and [6], our method reduces the effort associated with each distance computation (by decreasing the number of features involved in the calculation) without changing the total number of distance computations.

The idea of reducing the computational effort by reducing the number of training samples is known as the prototype approach (for example, see [2][15], [17], and [34]). According to [34], the training samples used as prototypes may not only be *selected* but also *modified* to optimize the classification performance. In addition, the training examples may be *replaced* by multiple prototypes [2]. The methods for prototype generation include competitive learning [2], genetic algorithms [17], and generalized delta rules [15]. The algorithm of [15] derives the weight of each feature in addition to

a set of prototypes. The calculated feature weights improve the classification accuracy, but they do not have any effect on the computational complexity. Thus, the prototype-based methods do not attempt to further reduce the computational effort by eliminating irrelevant and redundant features. Consequently, the complexity ratio of a prototype-based classifier (see sub-section 3.3 above), can be simply calculated as a ratio between the number of prototypes J and the total number of training samples X:

$$R_c = J / X \qquad (6)$$

In our approach, we use each training sample as a prototype (which may be modified in the future), but the distance computation by the first-stage classifier is based on a subset of selected features only.

All the above-mentioned methods for improving the efficiency of the k-NN classifier do not use any classifier combination. The serial combination approaches discussed in [29] involve at least two different types of classifiers within the same system (decision trees, fuzzy rules, etc.). At each stage, the corresponding classifier is usually applied to a different set of extracted features. No automated selection of features to be used by one of the classifiers is mentioned by [29], which confirms the novelty of the approach proposed in this paper.

The idea of classifying different instances by different classifiers (the first or the second classifier in our serial combination) is similar to the *Dynamic Classifier Selection (DCS)* approach ([11][12]). However, the authors of [11] suggest that the best classifier for a given test pattern is the one maximizing the so-called *Classifier Local Accuracy (CLA)*, disregarding its computational complexity. If both classifiers were of the same type (e.g., k-NN), the DCS approach would choose the full feature-set classifier in almost all cases, because its accuracy tends to be higher (see sub-section 4.5 above). Our system, on the other hand, allows sacrificing a certain amount of accuracy for the sake of reducing the computational effort.

6 Conclusions

In this paper, we have presented a novel method of serial classifier combination, which is aimed at speeding up the classification process for the majority of "well-behaved" instances by using only a small subset of relevant features. Whenever a classifier is not confident enough of its decision, the next classifier, which requires more features, is activated. The method is applied to a two-stage

serial combination of *k*-Nearest Neighbor Classifiers. The features used by the first classifier are selected by three alternative methods: Relief, IFN, and IFN + Relief. The decision of the first classifier is accepted only if the number of nearest neighbors in the majority class is above a pre-defined threshold. The results of experiments on benchmark datasets have shown that the computational complexity can be reduced by 50% and more, while maintaining the accuracy level of a single-stage classifier or even improving it. The last result confirms the claim of [29] that the overall classification performance of a multi-stage system can exceed the performance of any of its individual layers only if classifiers appearing at the different layers use different feature spaces. The computational efficiency of the two-stage system vs. a single-stage classifier has been confirmed by the actual classification times on the testing sets of instances.

In our view, the proposed sequential approach to classifier combination is still far from being fully explored and exploited. The overall computational effort can be further reduced by selecting an optimal set of prototypes out of the training set. The majority threshold used by a two-stage system can be selected automatically, based on a user-defined and problem-specific function relating the importance of maximum recognition accuracy to the importance of minimum computational complexity. A simple extension of the two-stage architecture would be adding more stages in the classifier combination, each based on a larger set of features. The approach is definitely not limited to a *k*-NN Classifier and it can be applied to other classification methods, e.g., Bayes Classifier, neural networks, and decision trees. The method and its extensions can be applied to classifying online streams of instances, like clickstream data on the web.

Acknowledgment

This work was partially supported by the USF Center for Software Testing under grant 2108-004-00 and by the National Institute for Systems Test and Productivity at University of South Florida under the USA Space and Naval Warfare Systems Command Grant No. N00039-01-1-2248.

References

[1] Belkasim SO, Shridhar M, Ahmadi, M. Pattern Classification Using an Efficient KNNR. Pattern Recognition 1992; 25(10):1269-1274.

[2] Bezdek JC, Reichherzer TR, Lim GS, Attikiouzel Y. Multiple-Prototype Classifier Design. IEEE Transactions on Systems, Man, and Cybernetics 1998; 28 (1):67-78.

[3] Blake CL, Merz CJ. UCI Repository of machine learning databases [http://www.ics.uci.edu/~mlearn/MLRepository.html], 1998.

[4] Breiman L, Friedman JH, Olshen RA, Stone PJ, Classification and Regression Trees. Wadsworth, 1984.

[5] Cover TM, Elements of Information Theory. Wiley, 1991.

[6] Djouadi A, Bouktache E. A Fast Algorithm for the Nearest Neighbor Classifier. IEEE Transactions on Pattern Analysis and Machine Intelligence 1997; 19 (3):277-282.

[7] Duin RPW, Tax DMJ. Experiments with Classifier Combining Rules. In: Kittler J, Roli F (eds). Multiple Classifier Systems. Springer, 2000, pp. 16-29.

[8] Dunteman GH, Principal Components Analysis. Sage Publications, Inc., 1989.

[9] Fayyad U, and Irani K. Multi-Interval Discretization of Continuous-Valued Attributes for Classification Learning. In Proceedings of the 13th International Joint Conference on Artificial Intelligence, 1993, pp. 1022-1027.

[10] Friedman JH, and Tukey JW. A Projection Pursuit Algorithm for Exploratory Data Analysis. IEEE Transactions on Computers 1974; 23 (9):881-889.

[11] Giacinto G, Roli F. Dynamic Classifier Selection. In Proceedings of Multiple Classifier Systems, First International Workshop, 2000, pp. 177-189.

[12] Giacinto G, Roli F. A Theoretical Framework for Dynamic Classifier Selection. In Proceedings of 15th International Conference on Pattern Recognition, 2000, pp. 8-11.

[13] Hansen LK, and Salamon P. Neural Network Ensembles. IEEE Transactions on Pattern Analysis and Machine Intelligence 1990; 12 (10):993-1001.

[14] Ho TK. The Random Subspace Method for Constructing Decision Forests. IEEE Transactions on Pattern Analysis and Machine Intelligence 1998; 20 (8):832-844.

[15] Huang YS, Chiang CC, Shieh J, Grimson E. Constructing Optimized Prototypes for Nearest Neighbor Classifiers. In Proceedings of the 15th International Conference on Pattern Recognition, 2000, pp. 17-20.

[16] John GH, Kohavi R, and Pfleger K. Irrelevant Features and the Subset Selection Problem. In Proceedings of the 11th International Conference on Machine Learning, 1994, pp. 121-129.

[17] Kangas J. Comparison Between two Prototype Representation Schemes for a Nearest Neighbor Classifier. In Proceedings of the 15th International Conference on Pattern Recognition, 2000, pp. 777-780.

[18] Kim J-O, and Mueller CW. Factor Analysis: Statistical Methods and Practical Issues. Sage Publications, Inc., 1978.

[19] Kira K, & Rendell LA. The Feature Selection Problem: Traditional Methods and a New Algorithm, In Proceedings of the Ninth National Conference on Artificial Intelligence, 1992, pp. 129-134.

[20] Kittler J, Hatef M, Duin RPW, and Matas J. On Combining Classifiers. IEEE Transactions on Pattern Analysis and Machine Intelligence 1998; 20 (3): 226-239.

[21] Kittler J, Roli F (eds). Multiple Classifier Systems, First International Workshop 2000. Springer.

[22] Lam L, Huang Y-S, and Suen CY. Combination of Multiple Classifier Decisions for Optical Character Recognition. In: Bunke H, and Wang PSP (eds). Handbook of Character Recognition and Document Image Analysis. World Scientific, 1997, pp. 79-101.

[23] Last M, Kandel A, Maimon O. Information-Theoretic Algorithm for Feature Selection. Pattern Recognition Letters 2001; 22 (6): 799-811.

[24] Liu H, and Motoda H. Feature Selection for Knowledge Discovery and Data Mining. Kluwer Academic Publishers, 1998.

[25] Maimon O, and Last M. Knowledge Discovery and Data Mining, The Info-Fuzzy Network (IFN) Methodology. Kluwer Academic Publishers, 2000.

[26] Mitchell TM. Machine Learning. McGraw-Hill, 1997.

[27] Pudil P, Novovociva P, Kittler J. Floating Search Methods in Feature Selection. Pattern Recognition Letters 1994; 15: 1119-1125.

[28] Quinlan JR. C4.5: Programs for Machine Learning. Morgan Kaufmann, 1993.

[29] Rahman AFR, and Fairhurst MC. Serial Combination of Multiple Experts: A Unified Evaluation. Pattern Analysis and Applications 1999; 2: 292-311.

[30] Schapire RE. A Brief Introduction to Boosting. In Proceedings of the 16th International Joint Conference on Artificial Intelligence, 1999, pp. 1-6.

[31] Schurmann J. Pattern Classification, A Unified View of Statistical and Neural Approaches. Wiley, 1996.

[32] Somol P, Pudil P, Novovicova J, Paclik P. Adaptive Floating Search Methods in Feature Selection, Pattern Recognition Letters 1999; 20: 1157-1163.

[33] Webb A. Statistical Pattern Recognition. Oxford University Press, 1999.

[34] Yan H. Prototype Optimization for Nearest Neighbor Classifiers Using a Two Layer Perceptron. Pattern Recognition 1993; 26 (2): 317-324.

Figures

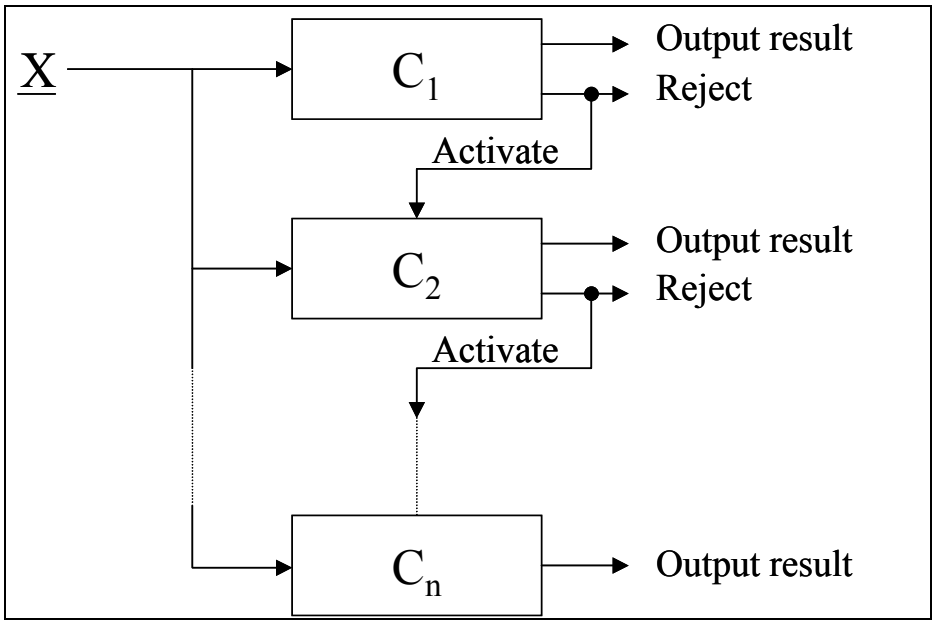


Figure 1 Proposed Classifier Combination Schema

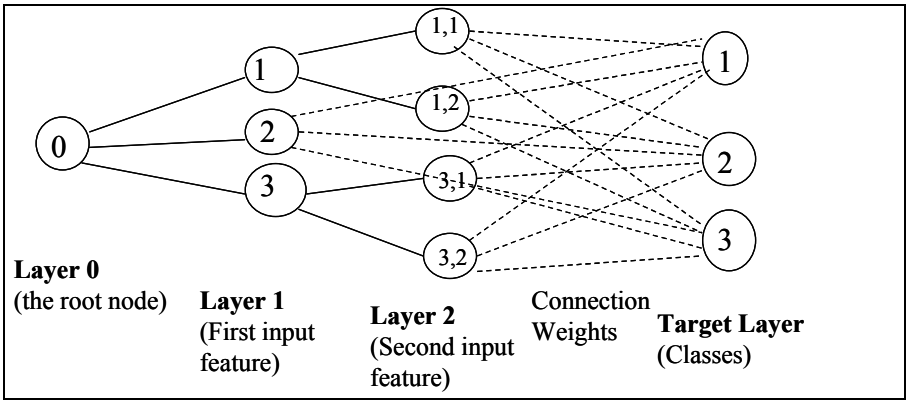


Figure 2 Information-Theoretic Connectionist Network - Two-Layered Structure

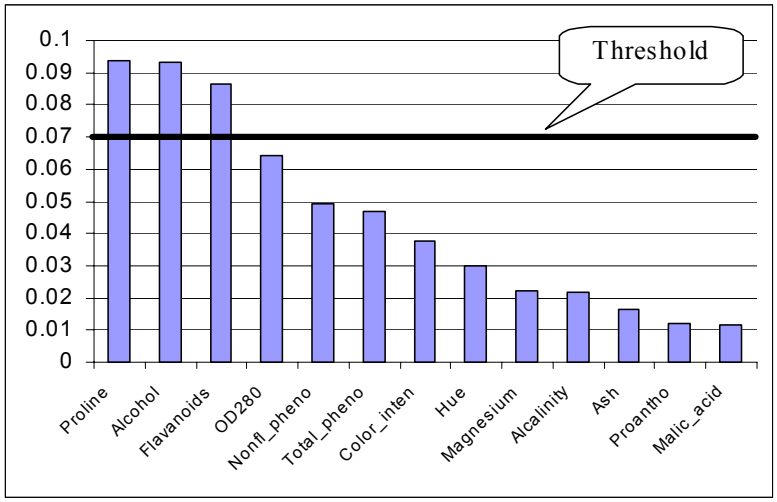


Figure 3 Relevance Levels: Wine Dataset

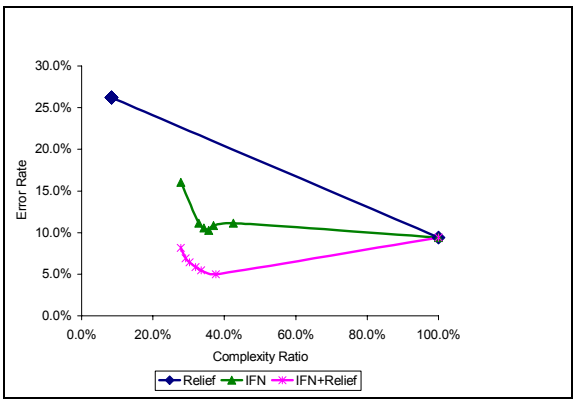


Figure 4 Chess Dataset: ROC

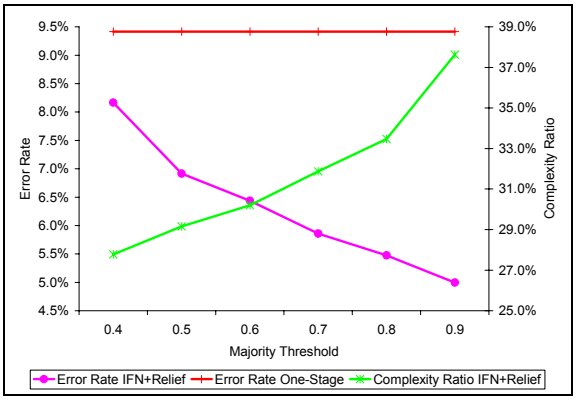


Figure 5 Chess Dataset: IFN + Relief

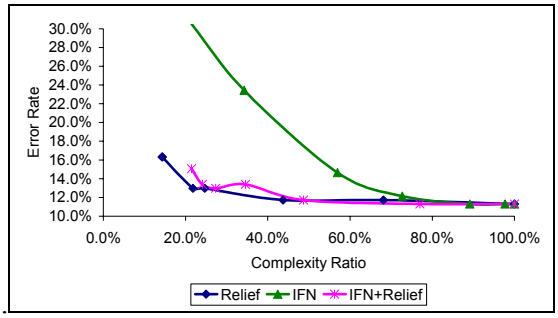


Figure 6 Credit Dataset: ROC

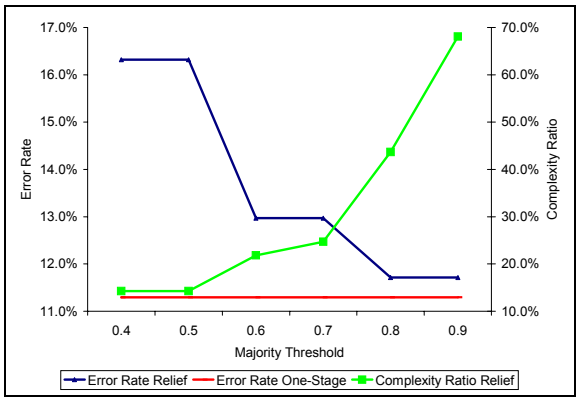
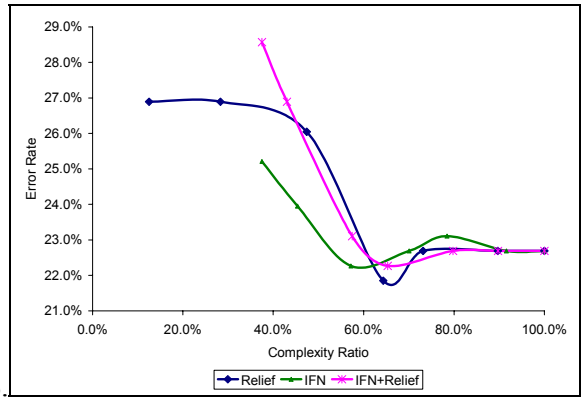


Figure 7 Credit Dataset: Relief



%.

Figure 8 Diabetes Dataset: ROC

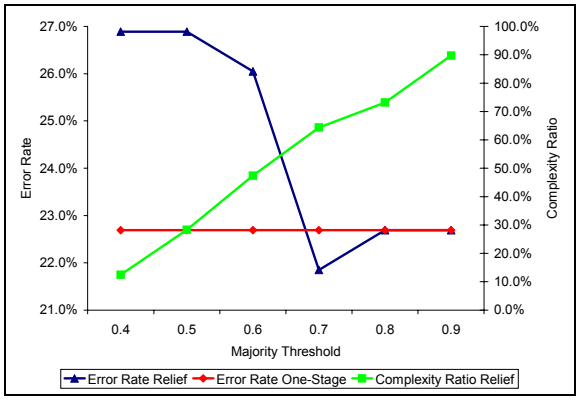


Figure 9 Diabetes Dataset: Relief

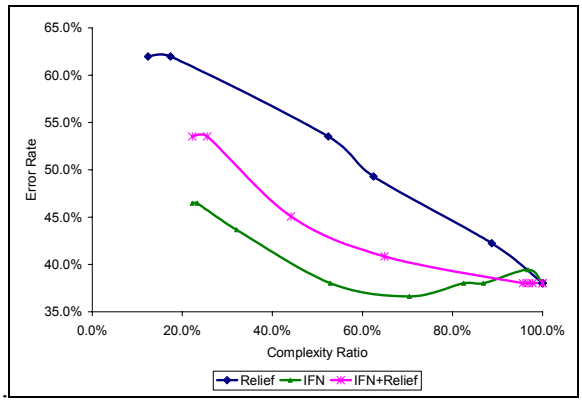


Figure 10 Glass Dataset: ROC

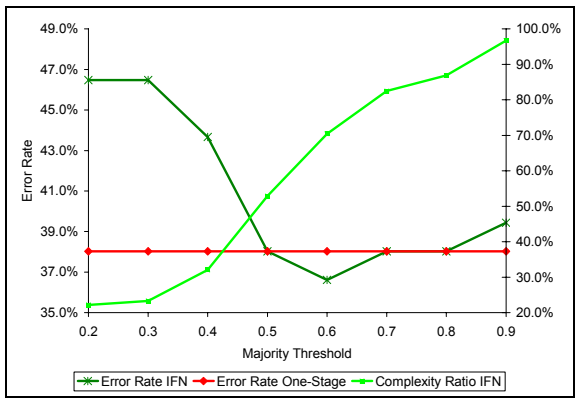


Figure 11 Glass Dataset: IFN

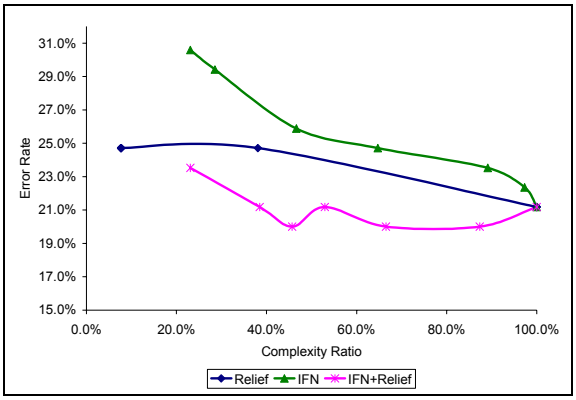


Figure 12 Heart Dataset: ROC

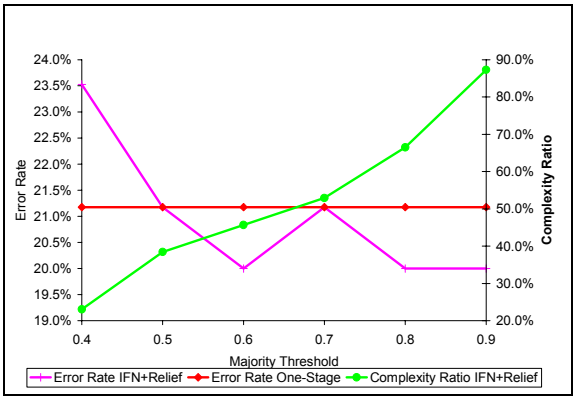


Figure 13 Heart Dataset: IFN + Relief

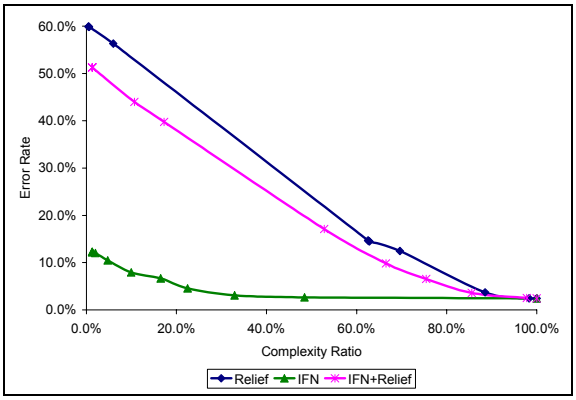


Figure 14 Opt_Orig Dataset: ROC

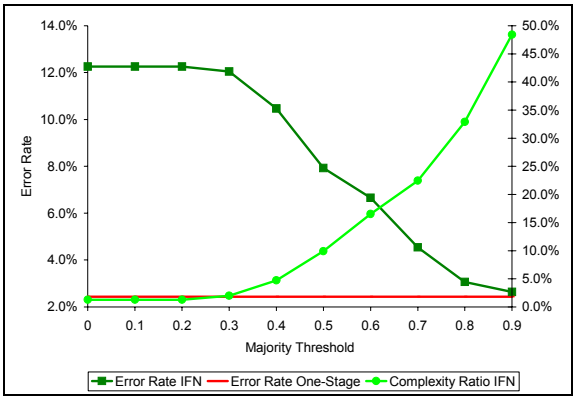


Figure 15 Opt_Orig Dataset: IFN

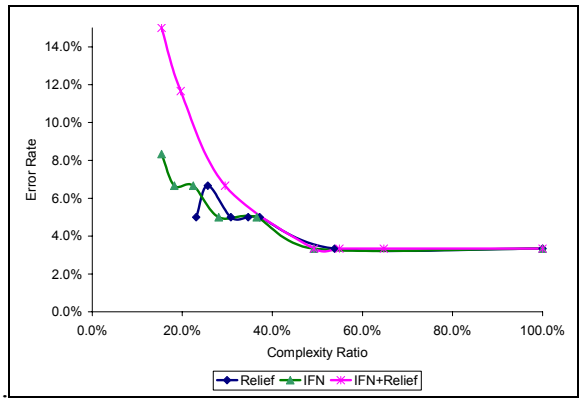


Figure 16 Wine Dataset: ROC

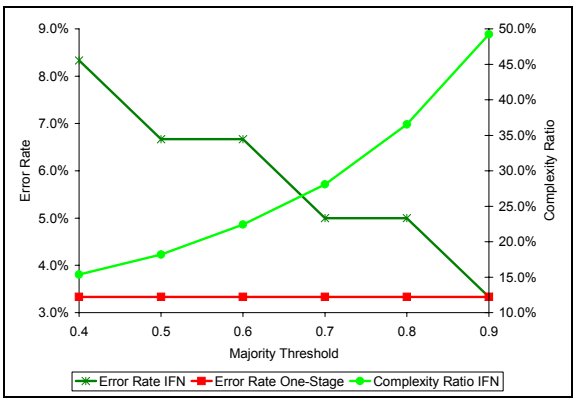


Figure 17 Wine Dataset: IFN

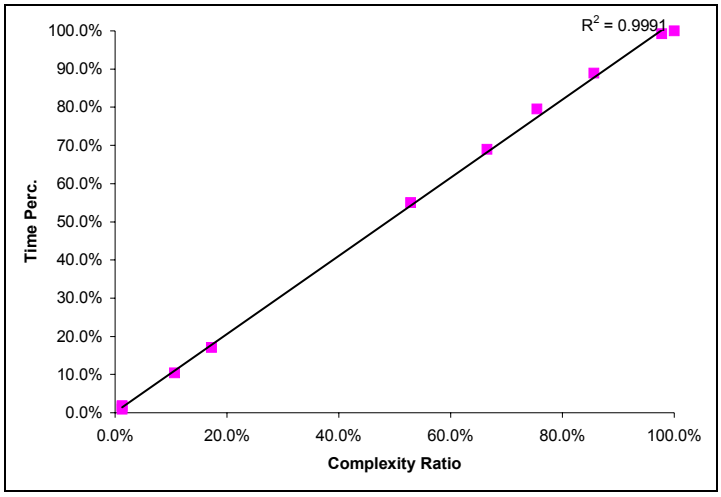


Figure 18 Opt_Orig Dataset: Time vs. Complexity Ratio

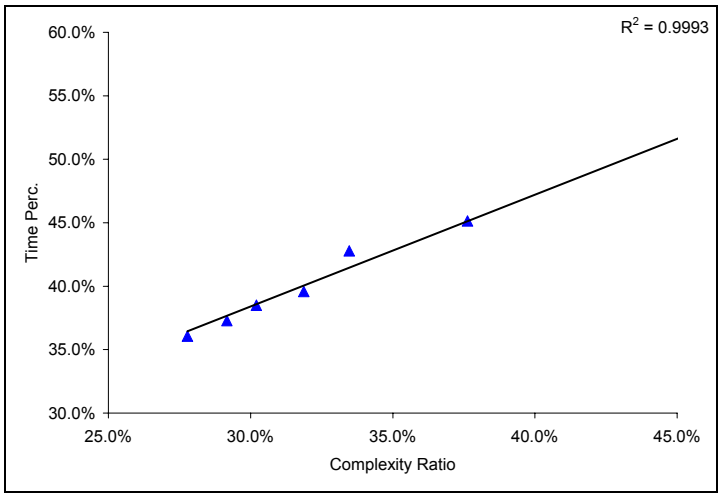


Figure 19 Chess Dataset: Time vs. Complexity Ratio

Tables

Table 1 Description of Datasets

					Number of Features		
Dataset	**Size**	**Training set**	**Testing set**	**Classes**	**Continuous**	**Nominal**	**Total**
Chess	3196	2155	1041	2	0	36	36
Credit	690	451	239	2	6	8	14
Diabetes	768	530	238	2	8	0	8
Glass	214	143	71	6	9	0	9
Heart	270	185	85	2	6	7	13
Optdigits-Orig	2880	1934	946	10	0	1024	1024
Wine	178	118	60	3	13	0	13

Table 2 Comparison of Feature Selection Methods

	Relevancy Threshold	Selected Features		Perc. of all		Testing Error Rate			
Dataset	**Relief**	**Relief**	**IFN**	**Relief**	**IFN**	**All features**	**After Relief**	**After IFN**	**After IFN+Relief**
Chess	0.1	3	10	8.3%	27.8%	0.094	0.262*	0.160*	0.082
Credit	0.1	2	3	14.3%	21.4%	0.113	0.163*	0.305*	0.151*
Diabetes	0.003	1	3	12.5%	37.5%	0.227	0.269	0.252	0.286*
Glass	0.1	1	2	11.1%	22.2%	0.380	0.620*	0.465	0.535*
Heart	0.2	1	3	7.7%	23.1%	0.212	0.247	0.306*	0.235
Optdigits-Orig	0.544	5	13	0.5%	1.3%	0.024	0.599*	0.123*	0.513*
Wine	0.07	3	2	23.1%	15.4%	0.033	0.050	0.083*	0.150*
Mean		**2.3**	**5.1**	**11.1%**	**21.2%**	**0.155**	**0.316**	**0.242**	**0.279**

Table 3 Comparison of Feature Selection Times

		Candidate Features			Feature Selection Time (sec.)	
Dataset	**Training set**	**Continuous**	**Nominal**	**Total**	**Relief**	**IFN**
Chess	2155	0	36	36	149.90	2.03
Credit	451	6	8	14	1.87	26.31
Diabetes	530	8	0	8	3.24	19.72
Glass	143	9	0	9	0.32	1.43
Heart	185	6	7	13	0.33	1.26
Optdigits-Orig	1934	0	1024	1024	339.16	115.02
Wine	118	13	0	13	0.17	0.88
Mean	**693**	**6**	**10**	**159.6**	**70.71**	**23.81**