Feature Extraction for Dynamic Integration of Classifiers

Mykola Pechenizkiy MPECHEN@CS.JYU.FI
Dept. of Mathematical Information Technology,
University of Jyväskylä,
Jyväskylä, Finland

Alexey Tsymbal ALEXEY.TSYMBAL@CS.TCD.IE
Dept. of Computer Science,
Trinity College Dublin,
Dublin, Ireland

Seppo Puuronen SEPI@CS.JYU.FI
Dept. of Computer Science and Information Systems,
University of Jyväskylä,
Jyväskylä, Finland

David W. Patterson WD.PATTERSON@ULSTER.AC.UK
Northern Ireland Knowledge Engineering Laboratory,
University of Ulster,
Belfast, U.K.

Abstract.

Recent research has shown the integration of multiple classifiers to be one of the most important directions in machine learning and data mining. In this paper, we present an algorithm for the dynamic integration of classifiers in the space of extracted features (FEDIC). It is based on the technique of dynamic integration, in which local accuracy estimates are calculated for each base classifier of an ensemble, in the neighborhood of a new instance to be processed. Generally, the whole space of original features is used to find the neighborhood of a new instance for local accuracy estimates in dynamic integration. However, when dynamic integration takes place in high dimensions the search for the neighborhood of a new instance is problematic, since the majority of space is empty and neighbors can in fact be located far from each other. Furthermore, when noisy or irrelevant features are present it is likely that also irrelevant neighbors will be associated with a test instance. In this paper, we propose to use feature extraction in order to cope with the curse of dimensionality in the dynamic integration of classifiers. We consider classical principal component analysis and two eigenvector-based class-conditional feature extraction methods that take into account class information. Experimental results show that, on some data sets, the use of FEDIC leads to significantly higher ensemble accuracies than the use of plain dynamic integration in the space of original features.

1 Introduction

Knowledge discovery in databases (KDD) is a combination of data warehousing, decision support, and data mining that provides an innovative approach to information management. It is widely regarded as the process of finding previously unknown and potentially interesting patterns and relations in large databases (Fayyad, 1996). Current electronic data repositories are growing quickly and contain a huge amount of data from commercial, scientific, and other domains. The capabilities for collecting and storing all kinds of data far exceed the ability to analyze, summarize, and extract knowledge from this data. Numerous data mining methods have been developed to extract knowledge from these large databases. Selection of the most appropriate data-mining method or a group of the most appropriate methods is usually not straightforward. Often the method of selection is done statically for all new instances of the domain area without analyzing each particular new instance. Usually better data mining results can be achieved if the selection method is done dynamically taking into account

characteristics of each new instance (Tsymbal, 2002).

Recent research has proved the benefits of the use of ensembles of base classifiers for classification problems (Dietterich, 1997). The challenge of integrating base classifiers is to decide which of them to select or how to combine their classifications to the final classification.

In many real-world applications, numerous features are used in an attempt to ensure accurate classification. If all those features are used to build classifiers, then they operate in high dimensions, and the learning process becomes computationally and analytically complicated. For instance, many classification techniques are based on Bayes decision theory or on nearest neighbor search, which suffer from the so-called "curse of dimensionality" (Bellman, 1961) leading to classification error in high dimensions (Aivazyan, 1989). Hence, there is a need to reduce the dimensionality of the feature space before classification. According to the adopted strategy dimensionality reduction techniques are divided into feature selection and feature transformation (also called feature discovery). The variants of the latter are feature extraction and feature construction. The key difference between feature selection and feature transformation is that during the former only a subset of original features is selected while the latter is based on the generation of completely new features; feature construction implies discovering missing information about the relationships among features by inferring or creating additional features (Liu, 1998). Feature extraction is a dimensionality reduction technique that extracts a subset of new features from the original set of features by means of some functional mapping, keeping as much information in the data as possible (Fukunaga, 1999).

An essential drawback of all methods that just assign weights to individual features is their insensitivity to interacting or correlated features. Also, in many cases some features are useful in one example set but useless or even misleading in another. That is why the transformation of the given representation before weighting features in such cases can be preferable. However, feature extraction and subset selection are of course not totally independent processes and they can be considered as different ways of a task representation. The use of such techniques is determined by the purposes and, moreover, sometimes feature extraction and selection methods are combined together in order to improve the solution.

In this paper, we consider the use of feature extraction in order to cope with the curse of dimensionality in the dynamic integration of classifiers. Feature extraction is focused on finding a more compact and better representation of instances to be used during the dynamic integration of classifiers for finding neighborhoods of a new instance. We propose the FEDIC (Feature Extraction for Dynamic Integration of Classifiers) algorithm, which combines a dynamic integration technique with a feature extraction technique.

Our hypothesis is that feature extraction affects classification accuracy in the same way (increase, decrease, or has no effect) with an ensemble and dynamic integration as with a single classifier (like *kNN*) for each individual data set. This assumption emerges from the idea that providing a better representation of instances, used during the dynamic integration process to find the neighborhood of a new instance, will result in more appropriate local estimates.

Particularly, in this paper we experiment with dynamic selection, dynamic voting and dynamic voting with selection integration techniques (Tsymbal *et al.*, 2001) and conventional PCA and two, class-conditional, eigenvector-based approaches (that use the within- and between-class covariance matrices and thus take class information into account when extracting features in contrast to PCA) that we used in our previous study (Tsymbal *et al*., 2002).

The paper is organized as follows. In the next section the dynamic integration of classifiers is discussed. Section 3 briefly considers PCA-based feature extraction techniques with respect to classification problems. In Section 4 we consider the FEDIC algorithm, which performs the dynamic integration of classifiers in the transformed space. In Section 5 experiments conducted on a number of data sets from the UCI machine learning repository are described, and the results of the FEDIC algorithm are analyzed and compared to the results of both the static and dynamic selection techniques shown in the non-transformed space.

2 Dynamic Integration of Classifiers

The integration of classifiers is an active research topic in machine learning, and different approaches have been considered (Chan, 1996). The integration of an ensemble of base classifiers has been shown to yield higher accuracy than the most accurate base classifiers alone with different real-world problems (Dietterich, 1997).

The task of using an ensemble of models can be broken down into two basic questions: (1) what set of learned models should be generated?; and (2) how should the predictions of the learned models be integrated? (Merz, 1998). To generate a set of accurate and diverse classifiers, and to integrate the predictions of classifiers several approaches have been tried. In the following subsections we consider some of these approaches.

2.1 Techniques for the generation of base classifiers of an ensemble

One way of generating a diverse set of models is to use learning algorithms with heterogeneous representations and search biases (Merz, 1998), such as decision trees, neural networks, instance-based learning, etc.

Another approach is to use models with homogeneous representations that differ in their method of search or in the data on which they are trained. This approach includes several techniques for generating base models, such as learning base models from different subsets of the training data. For example, two well-known ensemble methods of this type are bagging and boosting (Quinlan, 1996).

The base models with homogeneous representations may be binary classifiers that are integrated to implement a multiclass learner (i.e., where the number of class labels is greater than 2). Each classifier in such an ensemble is learnt to distinguish one class label from the others. For example, Dietterich and Bakiri (1995) map each class label onto a bit string prior to learning. Bit strings for class labels are designed to be well separated, thus serving as error-correcting output codes (ECOC). An off-the-shelf system for learning binary classifications (e.g., 0 or 1) can be used to build multiple classifiers, one for each bit in the output code. An instance is classified by predicting each bit of its output code (i.e., label), and then classifying the instance as the label with the "closest" matching output code.

Also, natural randomisation in the process of model search (e.g., random weight setting in the backpropagation algorithm for training neural networks) can be used to build different models with homogeneous representation. The randomisation can also be injected artificially. For example, Heath (1996) used a randomised decision tree induction algorithm for this purpose, which generated different decision trees every time it was run.

Another way of building models with homogeneous representations, which proved to be effective, is the use of different subsets of features for each model. For example, Oza and Tumer (1999) build base classifiers on different feature subsets, where each feature subset includes features relevant for distinguishing one class label from the others (the number of base classifiers is equal to the number of classes). Finding a set of feature subsets for constructing an ensemble of accurate and diverse base models is also known as ensemble feature selection (Opitz, 1999). In this work we will follow this approach to generate base classifiers.

Sometimes, a combination of the techniques considered above can be useful in order to provide the desired characteristics of the generated models. For example, a combination of boosting and wagging (which is a kind of bagging technique) is considered by Webb (2000).

In addition to these general-purpose methods for generating a diverse ensemble of models, there are learning algorithm-specific techniques. For example, Opitz and Shavlik (1996) employ a genetic algorithm in backpropagation to search for a good population of neural network classifiers.

2.2 Techniques for integration of an ensemble of models

Brodley and Lane (1996) have shown that simply increasing coverage of an ensemble through diversity is not enough to insure increased prediction accuracy. If the integration method does not utilize the coverage, then no benefit arises from integrating multiple

classifiers. Thus, the diversity and coverage of an ensemble are not alone sufficient conditions for ensemble accuracy but a good integration method that will utilize the diversity of the base models is also needed.

The challenging problem of integration is to decide which one(s) of the classifiers to rely on or how to combine the results produced by the base classifiers. Techniques using two basic approaches have been suggested as a solution to the integration problem: (1) a combination approach, where the base classifiers produce their classifications and the final classification is composed using them; and (2) a selection approach, where one of the classifiers is selected and the final classification is the result produced by it.

Several effective techniques for the combination of classifiers have been proposed. One of the most popular and simplest techniques used to combine the results of the base classifiers, is simple voting (also called majority voting and select all majority (SAM)) (Bauer & Kohavi, 1999). In the voting technique, the classification of each base classifier is considered as an equally weighted vote for that particular classification. The classification that receives the biggest number of votes is selected as the final classification (ties are solved arbitrarily). Often, weighted voting is used: each vote receives a weight, which is usually proportional to the estimated generalization performance of the corresponding classifier. Weighted Voting (WV) works usually much better than simple majority voting (Bauer & Kohavi, 1999).

More sophisticated combination techniques include the SCANN method based on the correspondence analysis and using the nearest neighbor search in the correspondence analysis results (Merz, 1998, 1999); and techniques to combine minimal nearest neighbor classifiers within the stacked generalization framework (Skalak, 1997). Two effective classifier combination techniques based on stacked generalization called “arbiter” and “combiner” were presented by Chan (1996). Hierarchical classifier combination has also been considered. Experimental results of Chan and Stolfo (1996, 1997) showed that the hierarchical (multi-level) combination approach, where the dataset was distributed among a number of sites, was often able to sustain the same level of accuracy as a global classifier trained on the entire dataset.

A number of selection techniques have also been proposed to solve the integration problem. One of the most popular and simplest selection techniques is the cross-validation majority (CVM, we call it simply Static Selection, SS, in our experiments) (Schaffer, 1993). In CVM, the cross-validation accuracy for each base classifier is estimated using the training set, and then the classifier with the highest accuracy is selected (ties are solved using voting). More sophisticated selection approaches include estimation of the local accuracy of the base classifiers by considering errors made in instances with similar predictions (Merz, 1995) or learning a number of meta-level classifiers (“referees”) that predict whether the corresponding base classifiers are correct or not for new instances (each “referee” is a C4.5 tree that recognizes two classes) (Koppel & Engelson, 1996). Todorovski and Dzeroski (2000) trained a meta-level decision tree, which dynamically selected a base model to be applied to the considered instance, using the level of confidence of the base models in correctly classifying the instance. An interesting approach to optimal classifier selection based on approximate reducts was proposed in Wroblewski (2001). Particularly interesting are the studies on selecting optimal subsets of classifiers. For example, Hao et al. (2003) presents optimistic results on searching the space of classifier subsets (by means of genetic algorithm and sequential search methods) to find the optimal subset.

The approaches to classifier selection can be divided into two subsets: static and dynamic selection. The static approaches propose one “best” method for the whole data space, while the dynamic approaches take into account each new instance to be classified and its neighbourhood only. The CVM is an example of the static approach, while the other selection techniques considered above are examples of the dynamic approach.

Techniques for combining classifiers can be static or dynamic as well. For example, widely used weighted voting (Bauer & Kohavi, 1999) is a static approach. The weights for each base classifier’s vote do not depend on the instance to be classified. In contrast, the reliability-based weighted voting (RBWV) introduced by Cordella (1999) is a dynamic voting

approach. It uses a classifier-dependent estimation of the reliability of predictions for each particular instance. Usually, better results can be achieved if the classifier integration is done dynamically taking into account the characteristics of each new instance.

2.3 Motivation for dynamic integration

Figure 1 presents a simple example of the distribution of errors of a model built by a C4.5 decision tree learning algorithm (Quinlan, 1993) over the instance space. The classification problem includes two features x_1 and x_2, and two classes "class_1" and "class_2". The target function is $x_1 > x_2$. The grey areas represent instances, which are incorrectly classified by the model represented by the broken line. It can be seen that the errors are concentrated in triangular regions in this case.

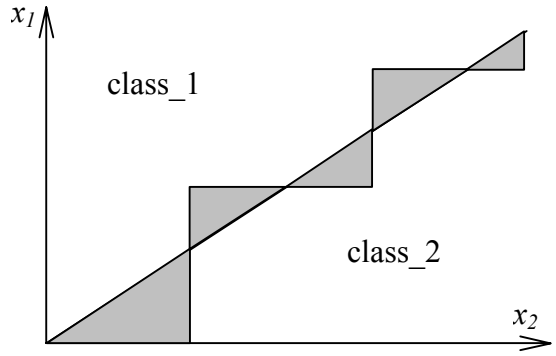

Figure 1 - An example of the distribution of errors of a C4.5 decision tree

Gama (1999) showed that for many types of classifiers the distribution of the error rate over the instance space is not homogeneous. Depending on the classifier, the error rate will be concentrated more in certain regions of the instance space than in others.

The basic idea of *dynamic integration* is that the information about a model's errors in the instance space can be used for learning just as the original instances were used for learning the model. Giacinto and Roli (1999) presented a theoretical framework of dynamic classifier selection showing that the accuracy of the dynamic selection technique, approaches the accuracy of the optimal Bayesian classifier as the number of instances in the dataset grows.

In this paper, a dynamic integration approach is presented that estimates the local accuracy of the base classifiers by analyzing their accuracy on nearby instances to the instance to be classified (Puuronen *et al.*, 1999a). Instead of directly applying selection or combination as an integration method, we use cross validation to collect information about the classification accuracies of the base classifiers and use this information to estimate the local classification accuracies for each new instance. These estimates are based on the weighted nearest neighbor classification (WNN) (Cost & Salzberg, 1993).

The proposed dynamic integration technique contains two main phases (Puuronen *et al.*, 1999a; Puuronen & Tsymbal, 2001). The general idea of the proposed dynamic integration technique, can be seen in Figure 2.

First, in the learning phase, the training set is partitioned into folds. During the cross validation run, we estimate the local classification errors of each base classifier for each instance of the training set according to the 1/0 loss function. These local errors together with the features of the instances of the training set form a meta-level training set used by WNN. The learning phase finishes with training the base classifiers on the whole training set. The application phase begins with determining the K-nearest neighborhood for the new instance using a distance metric based on the values of its features. Then, the meta-level classifier (WNN) is used to predict the local classification errors of each base classifier for a new instance using the meta-level training set. In WNN, to weight the classification errors for an instance k from the K-nearest neighborhood of the test instance, we use a tri-cube function (Hastie & Tibshirani, 1996):

$$w_k = (1 - (d_k / d_{K+1})^3)^3, \quad (1)$$

where d_k is the distance from the instance k to the test instance, and d_{K+1} is the distance from the test instance to the first nearest instance not included in the K-neighbourhood.

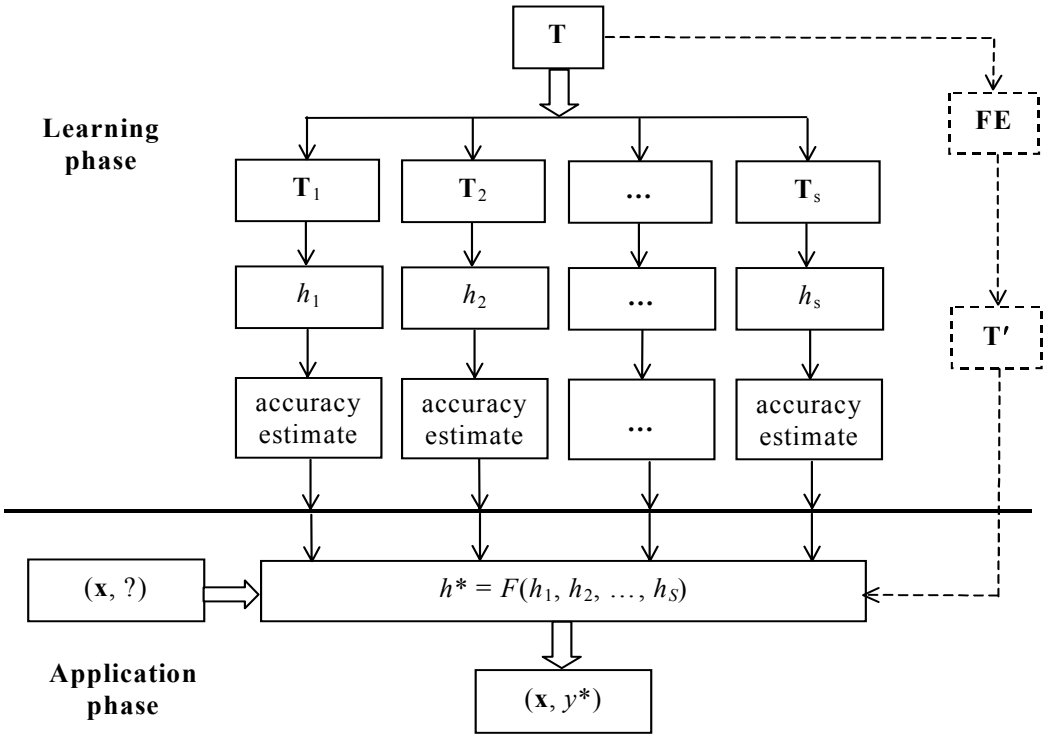

Figure 2 – Dynamic integration: learning and application phases

Three different approaches based on the local accuracy estimates have been proposed: Dynamic Selection (DS) (Puuronen *et al.*, 1999a), Dynamic Voting (DV) (Puuronen *et al.*, 1999a), and Dynamic Voting with Selection (DVS) (Tsymbal *et al.*, 2001). All these are based on the same local accuracy estimates obtained using WNN and the above weighting formula (1). DS simply selects a classifier with the least predicted local classification error, as was also proposed by Giacinto and Roli (1999) and Woods *et al.* (1997). In DV, each base classifier receives a weight that is proportional to the estimated local accuracy of the base classifier, and the final classification is produced by combining the votes of each classifier with their weights. In DVS, the base classifiers with the highest local classification errors are discarded (the classifiers with errors that fall into the upper half of the error interval of the base classifiers) and locally weighted voting (DV) is applied to the remaining base classifiers.

In this paper we consider the feature extraction process (depicted by dotted lines in Fig. 2) as a means of *neighborhood enhancement* with respect to the dynamic integration of classifiers during the application phase. This idea will be considered further in Section 4. In the next section we consider the basic ideas of PCA-based feature extraction for classification.

3 PCA-based Feature Extraction for Classification

In this section we consider the main idea of PCA-based feature transformation. Generally, PCA-based Feature Extraction for Classification can be seen as a search process among all possible linear transformations of the original feature set for the best one, which preserves class separability as much as possible in the space with the lowest possible dimensionality (Aladjem, 1994). In other words we are interested in finding a projection w:

$$\mathbf{y} = \mathbf{w}^T \mathbf{x} \tag{2}$$

where $\mathbf{y}$ is a $k \times 1$ transformed data point (presented using k features), $\mathbf{w}$ is a $d \times k$ transformation matrix, and $\mathbf{x}$ is a $d \times 1$ original data point (presented using d features).

The conventional PCA-based approach does not take into account class information. We consider a simple example that shows both good and bad performance of PCA from a class

discrimination point of view.

Then, we introduce class conditional feature extraction approaches that are based on Fisher linear discriminant analysis. The first approach is parametric in nature and the second approach is nonparametric.

In any case, all feature extraction methods have a similar property: they extract new features, which are different from the original ones, and thus constitute a new representation space, which usually has a significantly smaller number of features.

3.1 PCA-based feature transformations

Principal Component Analysis (PCA) is a classical statistical method, which extracts a lower dimensional space by analyzing the covariance structure of multivariate statistical observations (William & Goldstein, 1998; Jolliffe, 1986).

The main idea behind PCA is to determine the features that explain as much of the total variation in the data as possible with as few of these features as possible. We are interested in PCA primarily as a widely used dimensionality reduction technique, although PCA is also used for example for the identification of the underlying variables, for visualization of multidimensional data, identification of groups of objects or outliers and for some other purposes (Jolliffe, 1986).

The computation of the PCA transformation matrix is based on the eigenvalue decomposition of the covariance matrix **S** (and therefore it is computationally rather expensive).

$$\mathbf{w} \leftarrow eig\_decomposition\left(\mathbf{S} = \sum_{i=1}^{n} (\mathbf{x}_i - \mathbf{m})(\mathbf{x}_i - \mathbf{m})^T \right) \tag{3}$$

where n is the number of instances, $\mathbf{x}_i$ is the i-th instance, and $\mathbf{m}$ is the mean vector of the input data.

Computation of the principal components can be presented with the following algorithm:

1. Calculate the covariance matrix **S** from the input data.
2. Compute the eigenvalues and eigenvectors of **S** and sort them in a descending order with respect to the eigenvalues.
3. Form the actual transition matrix by taking the predefined number of components (eigenvectors).
4. Finally, multiply the original feature space with the obtained transition matrix, which yields a lower- dimensional representation.

The necessary cumulative percentage of variance explained by the principal axes is used commonly as a threshold, which defines the number of components to be chosen.

The following list summarizes the properties of PCA that are interesting from the feature extraction point of view: variance maximization of the extracted features; the extracted features are uncorrelated; best linear approximation in the mean-square sense, the truncation error is the sum of the lower eigenvalues; and information that is contained in the extracted features is maximized (William & Goldstein, 1998). Besides the properties pointed above PCA has the following advantages: the model parameters can be computed directly from the data, e.g. by diagonalizing the sample covariance of the data; and compression and decompression are quite easy operations and require only matrix multiplication.

Although PCA has a number of advantages, there are some drawbacks. One of them is that PCA gives high weight to features with higher variabilities irrespective of whether they are useful for classification or not. In Figure 3, adopted from Oza and Tumer (1999), we can see why it is dangerous for class information not to be used. The first case shows a classic example of PCA, where the first principal component corresponds to the variable with the highest discriminating power along the first principal axis but the second figure gives us an opportunity to see that in fact the chosen component is not relevant to this problem.

In order to overcome this problem, some variations of PCA with local and nonlinear processing are used to improve dimensionality reduction (Kambhatla & Leen, 1997). In spite of certain benefits for classification problems of such methods, compared to global PCA, we

have to recognise that they do not directly use class information. Additionally, in some cases, it can be difficult to understand the set of new features, extracted by PCA, although this problem relates to every feature extraction method.

3.2 Class conditional feature extraction

Feature extraction for classification is a search among all possible transformations for the best one, which preserves class separability as much as possible in the space with the lowest possible dimensionality (Aladjem, 1994).

Although PCA is still probably the most popular feature extraction technique, it has a serious drawback, namely, giving high weights to features with higher variabilities irrespective of whether they are useful for classification or not. This may give rise to the situation where the chosen principal component corresponds to an attribute with the highest variability but has no discriminating power (as it was considered in Figure 3).

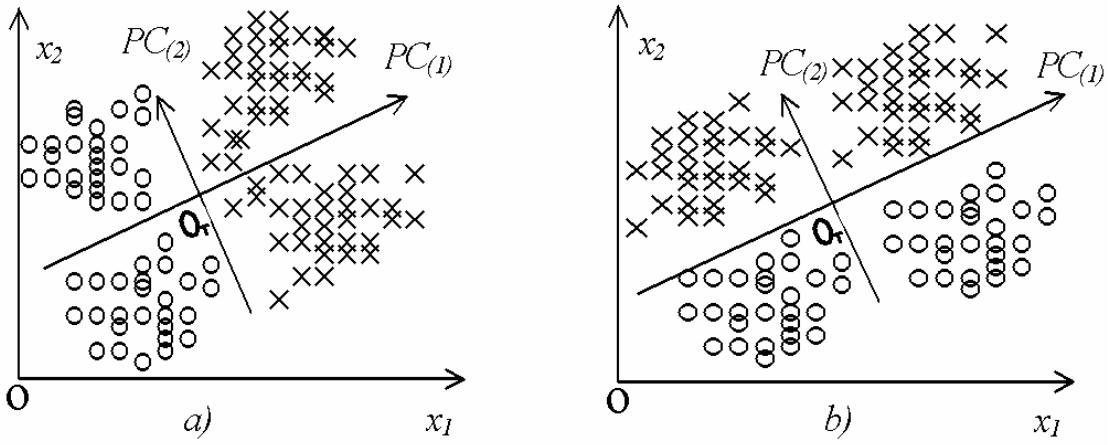

Figure 3 – PCA for classification: a) effective work of PCA, b) the case where an irrelevant principal component was chosen from the classification point of view (**O** denotes the origin of the initial feature space x_1, x_2 and $\mathbf{O_T}$ – the origin of the transformed feature space $PC_{(1)}$, $PC_{(2)}$)**.**

The usual decision is to use some class separability criterion, based on a family of functions of scatter matrices: the within-class covariance, the between-class covariance, and the total covariance matrices.

The within-class covariance matrix shows the scatter of samples around their respective class expected vectors:

$$\mathbf{S}_W = \sum_{i=1}^{c} n_i \sum_{j=1}^{n_i} (\mathbf{x}_j^{(i)} - \mathbf{m}^{(i)})(\mathbf{x}_j^{(i)} - \mathbf{m}^{(i)})^T \,, \tag{4}$$

where c is the number of classes, n_i is the number of instances in a class i, $\mathbf{x}_j^{(i)}$ is the j-th instance of i-th class, and $\mathbf{m}^{(i)}$ is the mean vector of the instances of the i-th class.

The between-class covariance matrix shows the scatter of the expected vectors around the mixture mean:

$$\mathbf{S}_B = \sum_{i=1}^{c} n_i (\mathbf{m}^{(i)} - \mathbf{m})(\mathbf{m}^{(i)} - \mathbf{m})^T \,, \tag{5}$$

where c is the number of classes, n_i is the number of instances in the class i, $\mathbf{m}^{(i)}$ is the mean vector of the instances of the i-th class, and $\mathbf{m}$ is the mean vector over all classes.

The total covariance matrix shows the scatter of all samples around the mixture mean. It can be shown analytically that this matrix is equal to the sum of the within-class and between-class covariance matrices (Fukunaga 1990):

$$\mathbf{S} = \mathbf{S}_B + \mathbf{S}_W \,. \tag{6}$$

One possible criterion based on the family of functions of the considered scatter matrices is Fisher linear discriminant:

$$J(\mathbf{a}) = \frac{\mathbf{a}^T \mathbf{S}_B \mathbf{a}}{\mathbf{a}^T \mathbf{S}_W \mathbf{a}}, \tag{7}$$

A number of other criteria has been proposed (Fukunaga, 1990). The criterion (7) can be optimized by the use of the *simultaneous diagonalization algorithm* (Fukunaga, 1990):

1. Transformation of $\mathbf{X}$ to $\mathbf{Y}$: $\mathbf{Y} = \mathbf{\Lambda}^{-1/2}\mathbf{\Phi}^{\mathrm{T}}\mathbf{X}$, where $\mathbf{\Lambda}$ and $\mathbf{\Phi}$ are the eigenvalues and eigenvectors matrices of $\mathbf{S}_W$.
2. Computation of $\mathbf{S}_B$ in the obtained $\mathbf{Y}$ space.
3. Selection of m eigenvectors of $\mathbf{S}_B$, $\boldsymbol{\psi}_1,...,\boldsymbol{\psi}_m$, which correspond to the m largest eigenvalues.
4. Finally, new feature space $\mathbf{Z} = \mathbf{\Psi}_m^T \mathbf{Y}$, where $\mathbf{\Psi} = [\boldsymbol{\psi}_1,...,\boldsymbol{\psi}_m]$, can be obtained.

3.3 Parametric vs. nonparametric feature extraction

In the previous subsection we introduced the main principle of class separability criterion, based on a family of functions of scatter matrices. In this subsection we consider the difference between parametric and nonparametric approaches for class conditional feature extraction.

It should be noticed that there is a fundamental problem with the parametric nature of the covariance matrices. The rank of the $\mathbf{S}_B$ is at most the *number of classes-1*, and hence no more than this number of new features can be obtained. The nonparametric method overcomes this problem by trying to increase the number of degrees of freedom in the between-class covariance matrix, measuring the between-class covariances on a local basis. The k-nearest neighbour (*kNN*) technique is used for this purpose.

A two-class nonparametric feature extraction method has been considered by Fukunaga (1990), and is extended by Tsymbal et al. (2002) to the multiclass case. The algorithm for nonparametric feature extraction is the same as for the parametric extraction. Simultaneous diagonalization is used as well, and the only difference is in calculating the between-class covariance matrix $\mathbf{S}_B$. In the nonparametric case the between-class covariance matrix is calculated as the scatter of the samples around the expected vectors of other classes' instances in the neighborhood:

$$\mathbf{S}_B = \sum_{i=1}^{c} n_i \sum_{k=1}^{n_i} w_{ik} \sum_{\substack{j=1 \\ j \neq i}}^{c} (\mathbf{x}_k^{(i)} - \mathbf{m}_{ik*}^{(j)})(\mathbf{x}_k^{(i)} - \mathbf{m}_{ik*}^{(j)})^T, \tag{8}$$

where $\mathbf{m}_{ik*}^{(j)}$ is the mean vector of the *nNN* instances of the *j*-th class, which are nearest neighbors to $\mathbf{x}_k^{(i)}$. The number of nearest instances *nNN* is a parameter, which should be set in advance. The coefficient w_{ik} is a weighting coefficient, which shows importance of each summand in (8). The goal of this coefficient is to assign more weight to those elements of the matrix, which involve instances lying near the class boundaries thus being more important for classification. We generalize the two-class version of this coefficient proposed by Fukunaga (1990) to the multiclass case:

$$w_{ik} = \frac{\min_j \{d^{\alpha}(\mathbf{x}_k^{(i)}, \mathbf{x}_{nNN}^{(j)})\}}{\sum_{j=1}^{c} d^{\alpha}(\mathbf{x}_k^{(i)}, \mathbf{x}_{nNN}^{(j)})}, \tag{9}$$

where $d(\mathbf{x}_k^{(i)}, \mathbf{x}_{nNN}^{(j)})$ is the distance from $\mathbf{x}_k^{(i)}$ to its *nNN*-nearest neighbour of the class *j*, and α is a parameter which should be set in advance.

Two parameters (*nNN* and α) are used to assign more weight to those elements of the matrix, which involve instances lying near the class boundaries and thus being more important for classification. For two-class case Fukunaga (1990) set the parameter α to 1 and *nNN* to 3, but without any strict justification. Tsymbal et al. (2002) has shown that these parameters have different optimal values for each data set.

The difference between parametric and nonparametric approaches in $\mathbf{S}_B$ calculation is depicted in Figure 4. Solid lines depict distances between class means and the global mean in

the parametric approach and local distances between individuals, not belonging to the same class, while dashed lines depict distances within a class.

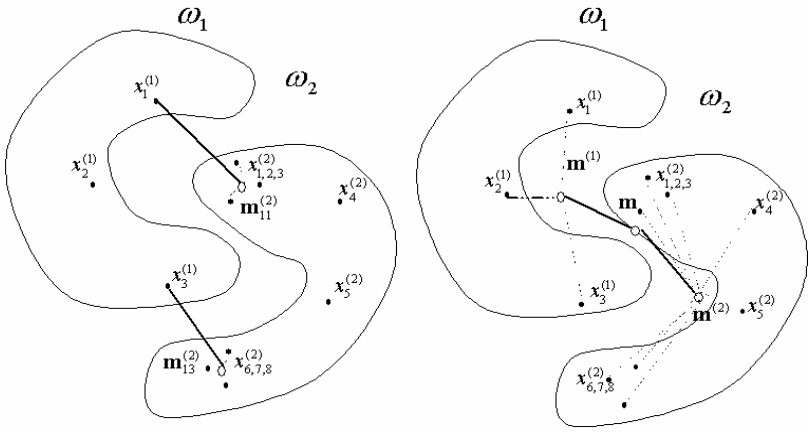

Figure 4 – Differences in the between-class covariance matrix calculation for nonparametric (left) and parametric (right) approaches for the two-class (ω_1 and ω_2) problem.

In the next section we will see how feature extraction is used in the dynamic integration of classifiers. A corresponding algorithm will be introduced.

4 Dynamic Integration of Classifiers with Instance Space Transformation

In order to address the curse of dimensionality in the dynamic integration of classifiers, we propose the FEDIC (Feature Extraction for Dynamic Integration of Classifiers) algorithm that first performs feature extraction and then uses a dynamic scheme to integrate classifiers.

4.1 Scheme of the FEDIC algorithm

In Figure 5, a scheme that illustrates the components of the FEDIC approach is presented. The FEDIC learning model consists of two major phases: *the training phase* and *the application phase*. The training phase is further divided into *the training of the base classifiers* phase, and *the feature extraction phase* (FE). The application phase is further divided into *the meta-learning phase* and *the dynamic integration phase* (DIC).

The model is built using a wrapper approach (Kohavi, 1995) where the variable parameters in FE and DIC can be adjusted to improve performance as measured at the model validation phase in an iterative manner. These parameters include the threshold value that is related to the amount of covered variance by the first principal components and thus defines the number of output features in the transformed space (it is set up for each feature extraction method); the optimal values of α and nNN parameters (as described in Section 3) in the nonparametric feature extraction technique and the number of nearest neighbors in DIC as described later.

The basic stages of the algorithm are shown in Figure 6. First, the data set is divided into the training set, the validation set, and the test set using stratified random sampling. The training set is duplicated into two copies. One copy of the training set is used for building the base classifiers, and the other is used during the feature extraction phase. The validation set is used in the refinement cycle during the training of base classifiers. The test set is used during the application phase.

First, the algorithm proceeds with the training of the base classifiers phase, and the feature extraction phase which can be performed independently (in parallel) from it. As a result of these phases the meta-data is constructed. The meta-data includes trained base classifiers and corresponding local accuracy estimates, transformation models and the transformed training set. As meta-data is now available the meta-learning phase can start.

A test instance is transformed by the same transformation process, so that it is in the same feature space as meta-data (transformed training set). As the result of this phase the estimates of the local accuracies of the base classifiers in the neighborhood of a test instance are produced. These local estimates are used in the dynamic integration of classifiers phase, where the corresponding integration procedure is applied and a final prediction is made.

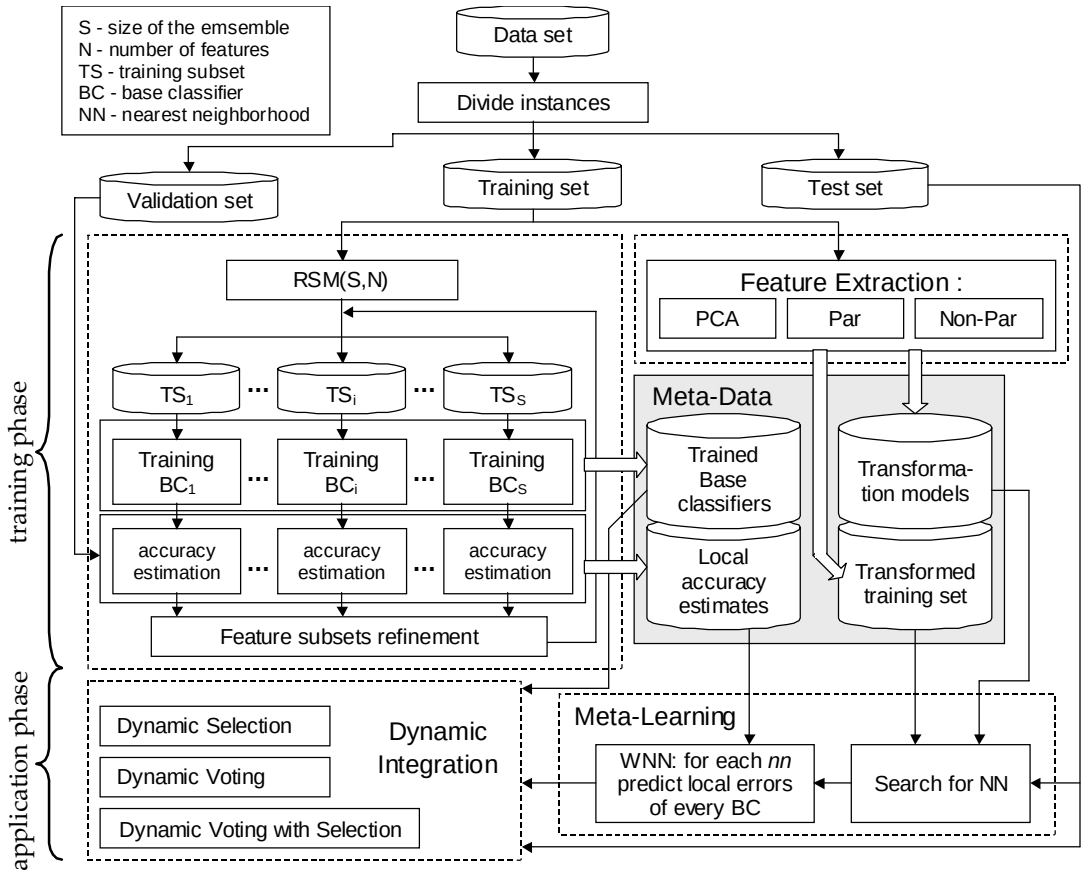

Figure 5 - Scheme of the FEDIC approach

```
D, T, V, Tst – whole data set, train, validation and test subsets
M = {L, BC, FE, T´} – meta-data
E – local error estimates
L – local accuracy estimates of BCs in the neighborhood of test instance
BC – set of base classifiers
FE = {PCA, Par, NPar} – transformation models
T´ – transformed train data sets
R – summary of results
thresh – percent of variance explained by extracted features selected
kNN, α – Non-Par settings

function FEDIC(D) returns R
         T,V,Tst ← divideDataSet(D)
         BC, E ← trainingBC(T,V)
         if (FE is NPar)
            set preselected_kNN_and_α
         FE, T´ ← featureExtraction(T, thresh, kNN, α)
         L ← metaLearning(E, FE, Tst, T´)
         return R ← DIC(Tst, L, BC)
```

Figure 6 – Main phases of the FEDIC algorithm

In the next subsections we further consider the training phase, the feature extraction phase, the meta-learning phase, and the dynamic integration of classifiers phase of the FEDIC algorithm.

4.2 Base classifiers training phase

The training phase (see Figure 7) consists of two main subphases: (1) construction of the initial ensemble in random subspaces; and (2) iterative refinement of the ensemble members. The iterative refinement, based on hill-climbing search, is used to improve the accuracy of the

base classifiers and the diversity of the ensemble. For all the feature subsets in turn each feature is tried to be imported or exported (included or deleted from the feature set). If the resulting new feature subset produces better performance on the validation set than the previous one then the change is kept. This process is continued until no further improvements are possible. The process usually takes no more than four passes through the feature set.

```
T   training set
V   validation set
BC  set of trained base classifiers
FS  set of feature subsets for the base classifiers
c(X) classification of instance X
Cj  j-th base classifier
Cj(X) prediction of Cj on instance X
Ej(X) estimation of error of Cj on instance X
E   error matrix
S   number of base classifiers
N   number of features

function trainingBC(T,V) returns BC, E
         FS = RSM (S, N) {the random subspace method}
         for i = 1 to S {refinement of each feature subset}
            FS[i] = hillClimbing(FS[i])
            Cj = trainBC(FS[i])
            for each X from V
               compare Cj(X) with c(X) and derive Ej(X)
               E ← Ei(X)
            BC ← Cj
         return BC, E
```

Figure 7 – Training the base classifiers

The feature subset's performance is measured using the fitness function proposed by Opitz (1999), where the fitness of a feature subset i has been selected to be proportional to the classification accuracy and diversity of the corresponding classifier:

$$Fitness_i = acc_i + \alpha \cdot div_i , \tag{9}$$

where acc_i and div_i are the accuracy and diversity calculated over the validation set, and α is the coefficient determining the degree with which the diversity influences the fitness of the current feature subset.

In general, measuring diversity is not that straightforward – there are a number of ways to measure diversity in ensembles of classifiers. In (Tsymbal et al., 2005) we considered five different pairwise measures of the ensemble diversity (plain disagreement, fail/non-fail disagreement, the Q statistic, the correlation coefficient, and the *kappa* statistic), which could be used as a component of the fitness function (9). In (Tsymbal et al., 2005) the best correlations of diversity measures (with the difference between the ensemble accuracy and the average base classifier accuracy) averaged over different data sets were shown by plain disagreement and fail/non-fail disagreement. In this study we use plain disagreement measure of diversity, which is equal to the proportion of the instances on which the classifiers make different predictions. So, for two classifiers i and j:

$$div_{i,j} = \frac{1}{N} \sum_{k=1}^{N} Diff(C_i(x_k), C_j(x_k)) , \tag{10}$$

where N is the number of instances in the data set, $C_i(x_k)$ is the class assigned by classifier i to instance k, and $Diff(a,b) = 0$ if $a = b$, otherwise $Diff(a,b) = 1$. The total ensemble diversity is the average of all the classifier pairs in the ensemble.

Tsymbal et al, (2003) have shown that the degree of importance of accuracy and diversity when building ensembles is different for different datasets. Consequently, the appropriate value of α in (10) is different for different data sets, and often accuracy significantly changes with different α. Obviously, the appropriate value of α can be selected by cross-validation during learning phase. However, the issue of "guessing" an optimal α setting for a data set at

consideration is still awaiting further research.

The training set **T** is partitioned into v folds. Then, cross-validation is used to estimate the errors of the base classifiers E_j(x*) on the training set.

The training phase continues with training the base classifiers C_j on the whole training set. As a result of this phase a set of trained base classifiers **BC** and the error matrix **E** are produced.

4.3 The feature extraction phase

The feature extraction phase begins with preprocessing which includes binarization of the categorical features. Each categorical feature is replaced with a redundant set of binary features, each corresponding to a value of the original feature. During this phase, feature extraction techniques are applied to the preprocessed training set **T** to produce transformed data with a reduced number of features. The pseudo-code of this process is shown in Figure 8.

T, **T**′ preprocessed train and transformed train data sets
S, $\mathbf{S_b}$, $\mathbf{S_w}$ total, between- and within-class covariance matrices
m mean vector
Y intermediate transformed space
$\mathbf{\Lambda}$, $\mathbf{\Phi}$ eigenvalues and eigenvectors matrices
threshold the amount of variance in the selected PCs

function PCA_FE(**T**, *threshold*) **returns** **T**′

$\mathbf{S} \leftarrow \sum_{i=1}^{n} (\mathbf{x}_i - \mathbf{m})(\mathbf{x}_i - \mathbf{m})^T$

$\mathbf{\Lambda}, \mathbf{\Phi} \leftarrow \text{eigs}(\mathbf{S})$ {the eigensystem decomposition}

$\mathbf{P_{PCA}} \leftarrow \text{formP}_{\text{PCA}}(\text{threshold}, \mathbf{\Lambda}, \mathbf{\Phi})$
{forms the transformation matrix}

return $\mathbf{T}' \leftarrow \mathbf{P_{PCA}}\mathbf{T}$

function Par_FE(**T**, *threshold*) **returns** **T**′

$\mathbf{Y} \leftarrow \text{getYspace}(\mathbf{T})$

$\mathbf{S}_B \leftarrow \sum_{i=1}^{c} n_i (\mathbf{m}^{(i)} - \mathbf{m})(\mathbf{m}^{(i)} - \mathbf{m})^T$ {computing of $\mathbf{S_b}$ in the **Y** space}

$\mathbf{\Lambda}, \mathbf{\Phi} \leftarrow \text{eigs}(\mathbf{S}_B)$

$\mathbf{P}_{Par} \leftarrow \text{formP}_{\text{Par}}(\text{threshold}, \mathbf{\Lambda}, \mathbf{\Phi})$

return $\mathbf{T}' \leftarrow \mathbf{P}_{Par}\mathbf{Y}$

function NPar_FE(**T**, *threshold*, kNN, α) **returns** **T**′

$\mathbf{Y} \leftarrow \text{getYspace}(\mathbf{T})$

$w_{ik} \leftarrow \dfrac{\min_j \{d^{\alpha}(\mathbf{x}_k^{(i)}, \mathbf{x}_{nNN}^{(j)})\}}{\sum_{j=1}^{c} d^{\alpha}(\mathbf{x}_k^{(i)}, \mathbf{x}_{nNN}^{(j)})}$

$\mathbf{S}_B \leftarrow \sum_{i=1}^{c} n_i \sum_{k=1}^{n_i} w_{ik} \sum_{\substack{j=1 \\ j \neq i}}^{c} (\mathbf{x}_k^{(i)} - \mathbf{m}_{ik*}^{(j)})(\mathbf{x}_k^{(i)} - \mathbf{m}_{ik*}^{(j)})^T$

$\mathbf{\Lambda}, \mathbf{\Phi} \leftarrow \text{eigs}(\mathbf{S}_B)$

$\mathbf{P}_{NPar} \leftarrow \text{formP}_{\text{NPar}}(\text{threshold}, \mathbf{\Lambda}, \mathbf{\Phi})$

return $\mathbf{T}' \leftarrow \mathbf{P}_{NPar}\mathbf{Y}$

function getYspace(**T**) **returns** **Y**

$\mathbf{S}_W \leftarrow \sum_{i=1}^{c} n_i \sum_{j=1}^{n_i} (\mathbf{x}_j^{(i)} - \mathbf{m}^{(i)})(\mathbf{x}_j^{(i)} - \mathbf{m}^{(i)})^T$

$\mathbf{\Lambda}, \mathbf{\Phi} \leftarrow \text{eigs}(\mathbf{S}_w)$

return $\mathbf{Y} \leftarrow \mathbf{\Lambda}^{-1/2}\mathbf{\Phi}^{\mathrm{T}}\mathbf{X}$

Figure 8 – The feature extraction phase

The pre-processed data set is the input for the FE module, which implements one of the three extraction function: (1) *PCA_FE* that implements conventional PCA, (2) *Par_FE* that implements parametric feature extraction, and (3) *NPar_FE* that implements nonparametric feature extraction. The function *getYspace* is used to calculate an intermediate transformed space needed for the parametric and nonparametric approaches.

4.4 The meta-learning phase

The instances of the test set **Tst**, transformed training set **T',** and local error estimates **E** form the input for the meta-learning phase (Figure 9) where the performance of each base classifier for a new test instance is predicted. A test instance undergoes the transformation process (a corresponding **FE** transformation model is used) so that it is in the same feature space as the meta-data (transformed training set). Then, the meta-learning phase proceeds, finding in the transformed training set **T**', a nearest neighbourhood, **NN,** of the transformed test instance **x'**.

```
x test instance from test set Tst
x' an instance x transformed with P_PCA, P_NPar or P_NPar
W  vector of weights for base classifiers
T' transformed training set
L_j(x) prediction of error of C_j on instance x
function metaLearning_phase(T', E, x, FE) returns L
        x' ← transformTst(x, FE)
        NN = FindNeighborhood(T', x', nn)
        loop for j from 1 to m
            L_j ← (1/nn) Σ_{i=1}^{nn} W_{NN_i} · E_j(x_{NN_i})  {WNN estimation}
          L ← L_j
        return L
```

Figure 9 – The meta-learning phase

The size *nn* of the set **NN** is an adjustable parameter. The classification error L_j is predicted for each base classifier C_j using the WNN procedure as described in Section 2.3. The result of this phase are the local accuracy estimates **L** of the base classifiers in the neighborhood of test instances.

4.5 The dynamic integration phase

The local accuracy estimates **L,** base classifiers **BC**, and test instances **Tst** form the input for the dynamic integration phase where the base classifier(s) is (are) selected to produce the classification for a new instance. One of the three approaches considered in Section 2.3 is used for this purpose: *DS_integration_phase, DV_integration_phase,* or *DVS_integration_phase* (Figure 10). The first function *DS_integration_phase* implements Dynamic Selection. In DS a classifier with the lowest error (with the least global error in the case of ties) is selected to make the final classification. The second function *DV_integration_phase* implements Dynamic Voting. In the DV application phase each base classifier C_j receives a weight W_j that depends on the local classifier's performance, and the final classification is conducted by voting classifier predictions $C_j(\mathbf{x})$ with their weights W_j. In the case of the *DVS_integration_phase*, the base classifiers C_j with highest local errors L_j are discarded (the classifiers with local errors that fall into the upper half of the error interval of the ensemble) and dynamic voting (DV) is applied to the restricted set of classifiers.

We do not devote a separate subsection to the model validation and model evaluation phases since they are performed in a straightforward way. At the validation phase, the performance of the given model with the given parameter settings on an independent validation data set is tested. The given model, its parameter settings and performance are recorded and at the final evaluation phase, the best model from the number of obtained

models is selected. This model is the one with best parameters set according the validation results. The selected model is then tested with a test data set.

```
x test instance from test set Tst
W   vector of weights for base classifiers
T original training set
L_j(x) prediction of error of C_j on test instance x

function DS_integration_phase(x, BC, L)returns class of x
         l ← argmin_j L_j  {number of cl-er with min. L_j }
         {with the least global error in the case of ties}
         return C_l(x)

function DV_integration_phase(x, BC, L)returns class of x
         W = 1 − L
         return Weighted_Voting(W, C_1(x), ..., C_m(x))

function DVS_integration_phase(x, BC, L)returns class of x
         threshold = ½ (upper L_j − lower L_j)
         list = selectBadCl-ers(threshold)
         discardClassifiers(list)
         return DV_integration_phase(x, BC, L)
```

Figure 10 – The dynamic integration phase

In the next section we consider our experiments where we analyzed and compared the above feature-extraction techniques with dynamic integration of classifiers.

5 Experimental Studies

In this section we describe the experimental settings and present results on 21 data sets with different characteristics taken from the UCI machine learning repository (Blake & Merz, 1998).

5.1 Experimental settings

For each data set 70 test runs were made. In each test run the data set was first split into the training set, the validation set, and the test set, by stratified random sampling. Each time 60 percent of the instances were included in the training set. The other 40 percent were divided into two sets of approximately equal size (the validation and test sets). The validation set was used in the iterative refinement of the ensemble. The test set was used for the final estimation of the ensemble accuracy.

To construct ensembles of base classifiers we have used the EFS_SBC (Ensemble Feature Selection for the Simple Bayesian Classification) algorithm, introduced in Tsymbal et al. (2003a). Initial base classifiers were built using Naïve Bayes on the training set and later refined using a hill-climbing cycle on the validation data set. The size of the ensemble was selected to be 25. It has been shown that the biggest gain is achieved already with this number of base classifiers (Bauer & Kohavi, 1999). The diversity coefficient α was selected as it was recommended by Tsymbal et al. (2003a) for each data set.

At each run of the algorithm, we collected accuracies for five types of integration methods for the base classifiers: Static Selection (SS), Weighted Voting (WV), Dynamic Selection (DS), Dynamic Voting (DV), and Dynamic Voting with Selection (DVS). In dynamic integration, the number of nearest neighbors for the local accuracy estimates was pre-selected from the set of six values: {1, 3, 7, 15, 31, 63} ($2^n - 1$, $n = 1,...,6$), for each data set separately. Heterogeneous Euclidean-Overlap Metric (HEOM) (Wilson & Martinez, 1997) was used for the calculation of the distances in dynamic integration. For each feature extraction technique PCA (conventional PCA), Par (parametric approach), and NPar (nonparametric approach)), we first considered experiments with the best eigenvalue threshold from the following set of ten values: 0.65, 0.75, 0.85, 0.9, 0.95, 0.97, 0.99, 0.995,

0.999, as was done in Tsymbal et al. (2002). However, then another ten values were used: 0.75, 0.85, 0.9, 0.95, 0.97, 0.99, 0.999, 0.99999, 0.9999999, 1, as these seemed to be more relevant from an algorithmic sensitivity point of view. All the possible settings used throughout the experimental study are summarized in Table 1.

Table 1. Settings of parameters in experiments

Feature extraction technique	PCA, Par, NPar
Threshold for every FE	0.75, 0.85, 0.9, 0.95, 0.97, 0.99, 0.999, 0.99999, 0.9999999, 1
kNN and α for NPar	are preselected for each data set according to Tsymbal et al. (2002), presented in Table 2
Integration method	SS, WV, DS, DV, DVS
k in DS, DV, DVS	1, 3, 7, 15, 31, or 63
diversity coefficient	are preselected for each data set according to Tsymbal et al. (2002) presented in Table 2

A multiplicative factor of 1 was used for the Laplace correction in simple Bayes. Numeric features were discretized into ten equal-length intervals (or one per observed value, whichever was less). Software for the experiments was implemented using the MLC++ machine learning library (Kohavi et al., 1996).

5.2 Experimental results on UCI data sets

The main characteristics of the 21 data sets, which include the name of a data set, the number of instances included in a data set, the number of different classes of instances, the number of different types of features (categorical and numerical) included in the instances, and pre-selected settings for *kNN*, *α* and *div_α* are presented in Table 2. In Tsymbal et al. (2001) results of experiments with feature subset selection techniques with the dynamic selection of classifiers using these data sets were presented.

Table 2. UCI data sets used in the study

Data set	Inst	Classes	Original features			Num-al + binarized cat-al	*kNN*	*α*	div α
			Cat-al	Num-al	Total				
Balance	625	3	4	0	4	20	255	1/3	2
Breast	286	2	9	0	9	48	1	5	1
Car	1728	4	6	0	6	21	63	5	2
Diabetes	768	2	0	8	8	8	127	1/5	1
Glass	214	6	0	9	9	9	1	1	4
Heart	270	2	0	13	13	13	31	1	1
Ionosphere	351	2	0	34	34	34	255	3	1
Iris Plants	150	3	0	4	4	4	31	1/5	2
LED	300	10	0	0	7	7	15	1/3	1
LED17	300	10	0	0	24	24	15	5	1/4
Liver	345	2	0	6	6	6	7	3	1
Lymphography	148	4	15	3	18	38	7	1	2
MONK-1	432	2	6	0	6	15	1	1	4
MONK-2	432	2	6	0	6	15	63	20	1/4
MONK-3	432	2	6	0	6	15	1	1/3	2
Soybean	47	4	0	35	35	35	3	1	1
Thyroid	215	3	0	5	5	5	215	3	2
Tic-Tac-Toe	958	2	9	0	9	27	1	1	4
Vehicle	846	4	0	18	18	18	3	3	4
Voting	435	2	16	0	16	48	15	1/3	1/2
Zoo	101	7	16	0	16	16	7	1/20	2

The basic accuracy results of the experiments with FEDIC algorithm on the 21 UCI data sets are presented in Appendix A (the results are averaged over 70 runs).

In Table 3, for the initial comparison of the results of FEDIC with the results of plain DIC when no feature extraction was used (Plain), we present the best results of dynamic

integration (from the dynamic selection, dynamic voting and dynamic voting with selection schemes) with respect to each feature extraction technique, namely PCA, parametric (Par) and nonparametric (NPar) approaches. In the same table we present for comparison the classification accuracies of the 3NN classifier applied together with the same feature extraction techniques and without feature extraction (Plain). The last column contains the best classification accuracies for the static integration of classifiers (SIC) selected from static selection and weighted voting. Each row of the table corresponds to a single data set. The last row includes the results averaged over all the data sets. The bold-faced and underlined accuracies represent the approaches that were significantly better than all the other approaches; the bold-faced only accuracies represent the approaches that were significantly worse on the corresponding data sets (according to the Student t-test with 0.95 level of significance), and the shaded italic accuracies highlights poor performance of parametric FE.

Table 3. Accuracy results of the 3NN classifier and FEDIC algorithm (the best from DS, DV, DVS is selected)

Data set	3-NN				FEDIC				SIC
	Plain	PCA	Par	NPar	Plain	PCA	Par	Npar	Plain
Balance	.834	.827	.893	.863	.898	.898	.897	.899	.896
Breast	.724	.721	.676	.676	.744	.747	*.731*	.747	.744
Car	**.806**	**.824**	**.968**	**.964**	**.911**	**.920**	**.941**	**.942**	**.863**
Diabetes	.730	.730	.725	.722	.763	.767	.761	.763	.761
Glass	.664	.659	*.577*	.598	.679	.674	*.603*	.621	.623
Heart	.790	.777	.806	.706	.839	.839	.838	.839	.839
Ionosphere	.849	.872	.843	.844	.918	.920	.915	.917	.916
Iris	.955	.963	.980	.980	.941	.933	.940	.935	.929
LED	.667	.646	.630	.635	.745	.746	.745	.745	.751
LED17	.378	.395	.493	.467	.690	.690	.690	.690	.690
Liver	**.616**	**.664**	**.612**	**.604**	**.625**	**.635**	**.621**	**.623**	**.615**
Lymph	.814	.813	.832	.827	.841	.840	.828	.836	.824
Monk-1	**.758**	**.767**	***.687***	**.952**	**.832**	**.838**	***.709***	**.942**	**.746**
Monk-2	**.504**	**.717**	**.654**	**.962**	**.665**	**.665**	**.663**	**.675**	**.664**
Monk-3	.843	.939	.990	.990	.984	.975	.985	.987	.971
Soybean	.995	.992	.987	.986	1	1	1	1	1
Thyroid	.938	.921	.942	.933	.961	.958	.951	.955	.953
Tic	**.684**	**.971**	**.977**	**.984**	**.930**	**.964**	***.783***	**.895**	**.730**
Vehicle	**.694**	**.753**	**.752**	**.778**	**.676**	**.717**	***.668***	**.717**	**.603**
Voting	.921	.923	.949	.946	.953	.953	.945	.949	.951
Zoo	.932	.937	.885	.888	.960	.961	.959	.961	.948
average	.766	.801	.803	.824	.836	.840	.818	.840	.810

From Table 3 one can see that the nonparametric approach has the best accuracy on average with the base classifier (3NN) and shares the best accuracy results with PCA for the dynamic integration approaches. The parametric approach extracted the least number of features (and it was the least time-consuming approach) but its performance was unstable. The parametric approach has rather weak results for dynamic integration on the *Breast, Glass, Monk-1,* and *Tic* and *Vehicle* data sets in comparison to the other feature extraction approaches. FEDIC, with parametric feature extraction, has the worst average accuracy (even worse than dynamic integration in the space of original features). This contradicts the results of the parametric approach for single 3NN classifier. It had poor results only on *Glass* and *Monk-1* data sets and was the second best approach. On the contrary, PCA was the worst (although the most stable) feature extraction approach for the 3NN classifier, but shows equally good results on average as the nonparametric approach when applied with dynamic integration. However, it can be seen that the behavior of the feature extraction approaches may differ from one data set to the other.

The results of Table 3 show that, in some cases, dynamic integration in the space of extracted features results in significantly higher accuracies than dynamic integration in the space of original features.

This is the situation with the *Car, Liver, Monk-1, Monk-2, Tic-Tac-Toe,* and *Vehicle* data sets. We highlight this with bold font in the table and present a histogram in Figure 11, where, for each data set we compare the best accuracy from {ds_plain, dv_plain, dvs_plain} marked *DIC* with best accuracy from {ds_pca, dv_pca, dvs_pca; ds_par, dv_par, dvs_par; ds_npar, dv_npar, dvs_npar} marked *FEDIC*. It can be seen that on *Car, Liver, Monk1-2, Tic-Tac-Toe*, and *Vehicle* data sets feature extraction for dynamic integration of classifiers results in a notable increase of accuracy compared to the performance of dynamic integration of classifiers in the space of original features. Thus, the corresponding average increase of accuracy for these data sets was 3.1% (.911 vs. .942), 1% (.625 vs. .635), 11% (.832 vs. .942), 1% (.665 vs. 675), 3.4% (.93 vs. .964), and 4.1% (.676 vs. .717) respectively.

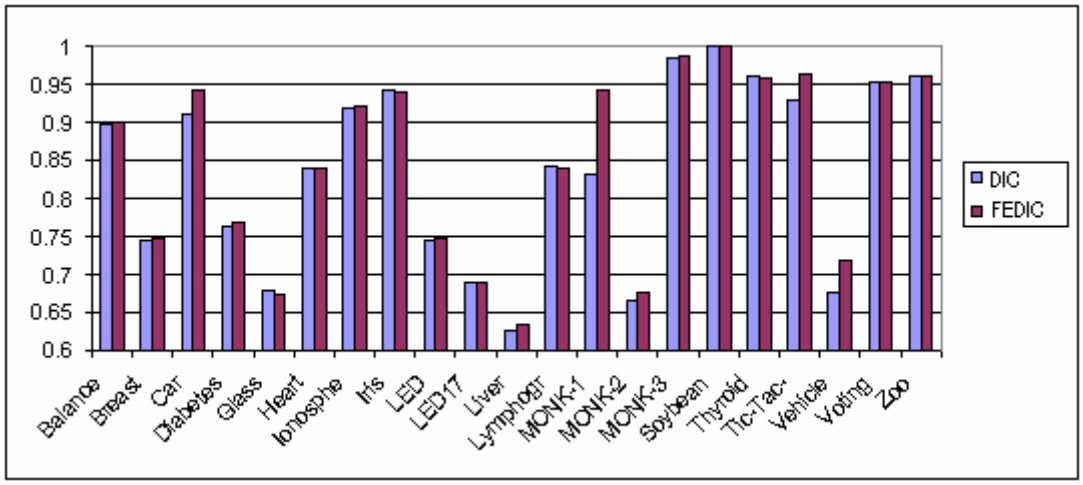

Figure 11 – Accuracy results on 21 UCI data sets: DIC vs. FEDIC

There is an important trend in the results – as a rule the FEDIC algorithm outperforms dynamic integration on plain features only on those data sets, on which feature extraction for classification with a single classifier provides better results than the classification on the plain features. If we analyze this correlation further (see Table 4), we will come to the conclusion

Table 4. Accuracy improvement (%) due FE for 3NN classifier (FE-plain), due DIC for base classifier (DIC-SIC), and due FE for DIC (FEDIC-DIC). The last column shows whether the difference is significant ('+' - significant, '–' not significant) according to the Student t-test with 0.95 level of significance.

Data set	FE – plain(%)	DIC – SIC (%)	FEDIC-DIC (%)	Significant
Balance	5.9	0.2	0.1	+ / – / –
Breast	-0.3	0.0	0.3	– / – / –
Car	16.2	4.8	3.1	+ / + / +
Diabetes	0.0	0.2	0.4	– / – / –
Glass	-0.5	5.6	-0.5	– / + / –
Heart	1.6	0.0	0.0	+ / – / –
Ionospher	2.3	0.2	0.2	+ / – / –
Iris	**2.5**	**1.2**	**-0.1**	**+ / + / –**
LED	-2.1	-0.6	0.1	– / – / –
LED17	11.5	0.0	0.0	+ / – / –
Liver	4.8	1.0	1.0	+ / + / +
Lymph	**1.8**	**1.7**	**-0.1**	**+ / + / –**
Monk-1	19.4	8.6	11.0	+ / + / +
Monk-2	**45.8**	**0.1**	**1.0**	**+ / – / +**
Monk-3	**14.7**	**1.3**	**0.3**	**+ / + / –**
Soybean	-0.3	0.0	0.0	– / – / –
Thyroid	0.4	0.8	-0.3	– / – / –
Tic	30	20.0	3.4	+ / + / +
Vehicle	8.4	7.3	4.1	+ / + / +
Voting	2.8	0.2	0.0	+ / – / –
Zoo	0.5	1.2	0.1	– / + / –
average	5.8	2.6	0.4	

that feature extraction influences the accuracy of dynamic integration to a similar extent as feature extraction influences the accuracy of base classifiers. This trend supports our expectations about the behavior of the FEDIC algorithm.

However, it could be seen also from Table 4 that this rule does not always hold. For example, with Iris dataset we have a situation where FE improves classification of *3*NN, and DIC outperforms static integration, nevertheless, FEDIC does not result in improvement of DIC in the original space. A similar situation is observed with *Lymph* and *Monk-3* datasets. The reason for this behavior is that both the meta-level learning process in dynamic integration, and the base learning process in training the base classifiers use the same feature space. Though, it is necessary to note, that the supervised information is different in those learning tasks (local classification errors for ensemble learning and the class labels themselves for learning a base classifier). Thus, the feature space is the same, and the output values to be predicted are different. This indicates that the influence of feature extraction on the accuracy of dynamic integration in comparison with its influence on the accuracy of a single classifier, is still different to a certain degree.

Still, the biggest revelation for us was that in one case, with the *Monk-2* dataset, FEDIC outperformed DIC in the original space (as FE with 3NN was better than plain *3*NN) even though DIC was not significantly better than static integration (see the corresponding row in Table 4).

The six data sets where FEDIC outperformed DIC were selected for further analysis. For these data sets we carried out a pairwise comparison of each FEDIC technique with the others and with static integration using the paired Student *t*-test at a 0.95 level of significance. Results of the comparisons are given in Table 5. Columns 2-6 of the table contain the results of comparing a technique corresponding to the row of a cell with a technique corresponding to the column, using the paired *t*-test. Each cell contains win/tie/loss information according to the *t*-test. For example, PCA has 3 wins against the parametric extraction, 1 draw and 1 loss on 5 data sets.

It can be seen from Table 5 that different FE methods may result in the better performance of dynamic integration of classifiers.

Table 5. Results of the paired *t*-test (win/tie/loss information) for six data sets, on which FEDIC outperforms plain dynamic integration

	PCA_DIC	Par_DIC	Nonpar_DIC	Plain_DIC	SIC
PCA_DIC		3/1/1	2/0/3	4/1/0	4/1/0
Par_DIC	1/1/3		0/2/3	2/2/1	3/1/1
Nonpar_DIC	3/0/2	3/2/0		4/1/0	5/0/0
Plain_DIC	0/1/4	1/2/2	0/1/4		1/3/1
SIC	0/1/4	1/1/3	0/0/5	1/3/1	

In Figure 12 we further analyse FEDIC accuracy results on the *Car, Liver, Monk-1, Monk-2, Tic-Tac-Toe* and *Vehicle* data sets. It can be seen that behaviour of the different approaches differ from one data set to another.

On the *Car* data set (Figure 12, a) in each case, the dynamic integration procedure in the space of extracted features is better than the corresponding dynamic integration procedure in the space of original features. Integration methods with parametric and nonparametric FE behave in a very similar way. However, these class conditional approaches give additional significant increase in accuracies compared to PCA. This means that PCA disregards class information, that seems to be important for this data set (as in Figure 3). DS and DVS work better than DV in both the original and extracted space. DS is better than DVS when applied in the original space and in the feature space extracted by PCA. However this difference is not significant for the feature spaces extracted by class conditional approaches.

The situation with *Liver* data set (Figure 12, b) is the opposite to the Car data set. Class conditional approaches do not improve the accuracies of dynamic integration. DS with the parametric and nonparametric FE show even significantly lower accuracies. PCA on the contrary significantly increases the accuracies: 1% for DV (the best integration method both in the original space and in the feature space extracted by PCA), 4% for DS and 3.5% for DVS.

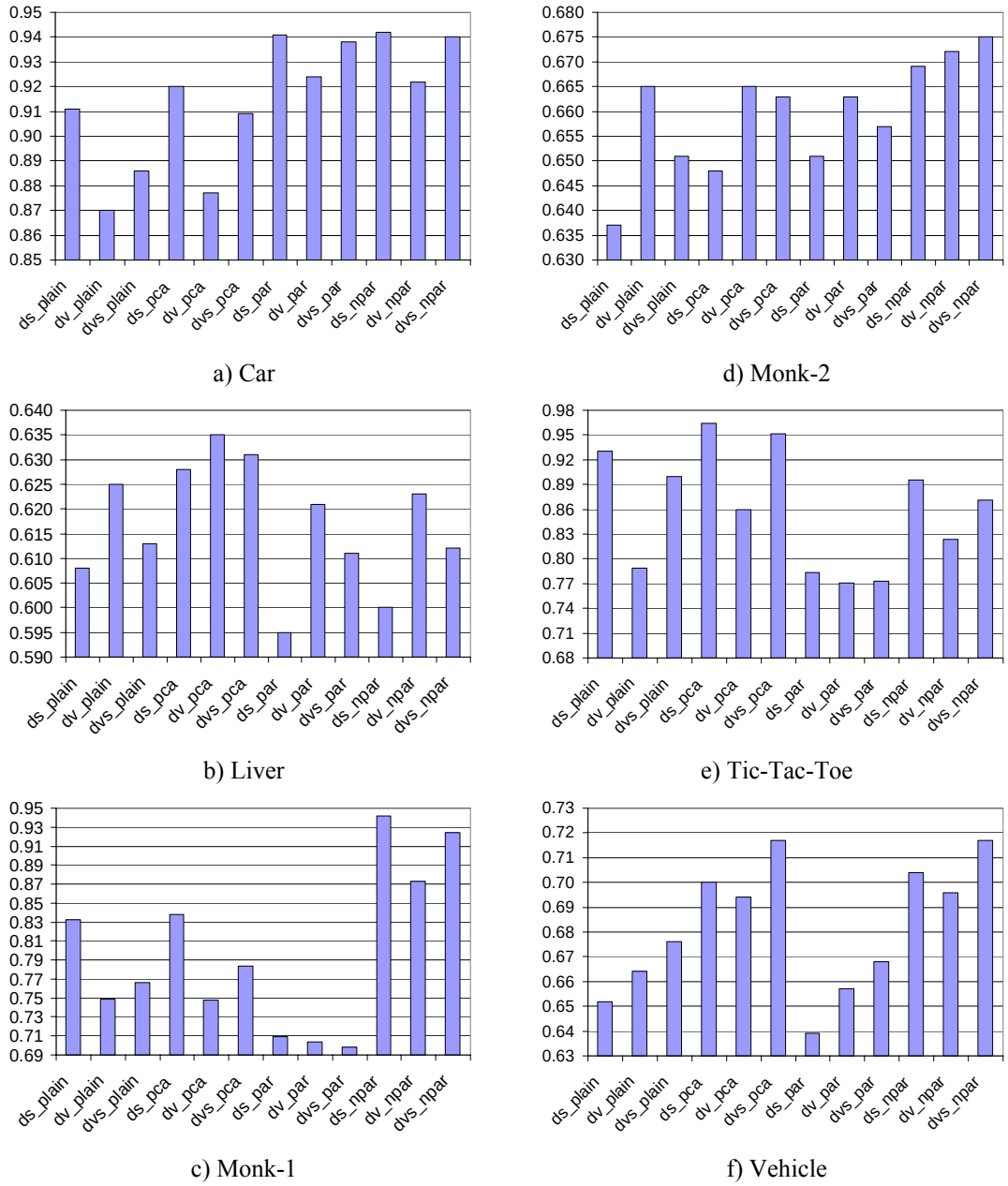

Figure 12 FEDIC accuracy results for Car, Liver, Monk-1, Monk-2, Tic-Tac-Toe and Vehicle data sets

For both *Monk-1* and *Monk-2* problems (Figure 12, c and d) the main increase of accuracies are achieved using nonparametric FE. The average corresponding increase in accuracies is about 10%. Parametric FE shows the worst results with any integration method, and PCA provides no significant change for dynamic integration.

The behaviour of feature extraction approaches on the *Tic-Tac-Toe* data set (Figure 12, e) is similar to its behavior on the *Liver* data set: class conditional approaches have a detrimental effect on dynamic integration, especially, parametric FE. And PCA, on the contrary, significantly increases the accuracies of every integration method.

On the *Vehicle* data set (Figure 12, f) PCA and nonparametric FE show similar results, but the parametric approach extracts too few features and fails – the results are even worse than those for DIC in the original space. DVS is the best integration method for this data set and together with PCA and nonparametric FE it increases accuracy 4%.

In Figure 13 we grouped the same accuracy results by integration method (a, b and c) and by feature extraction method (d, e, and f).

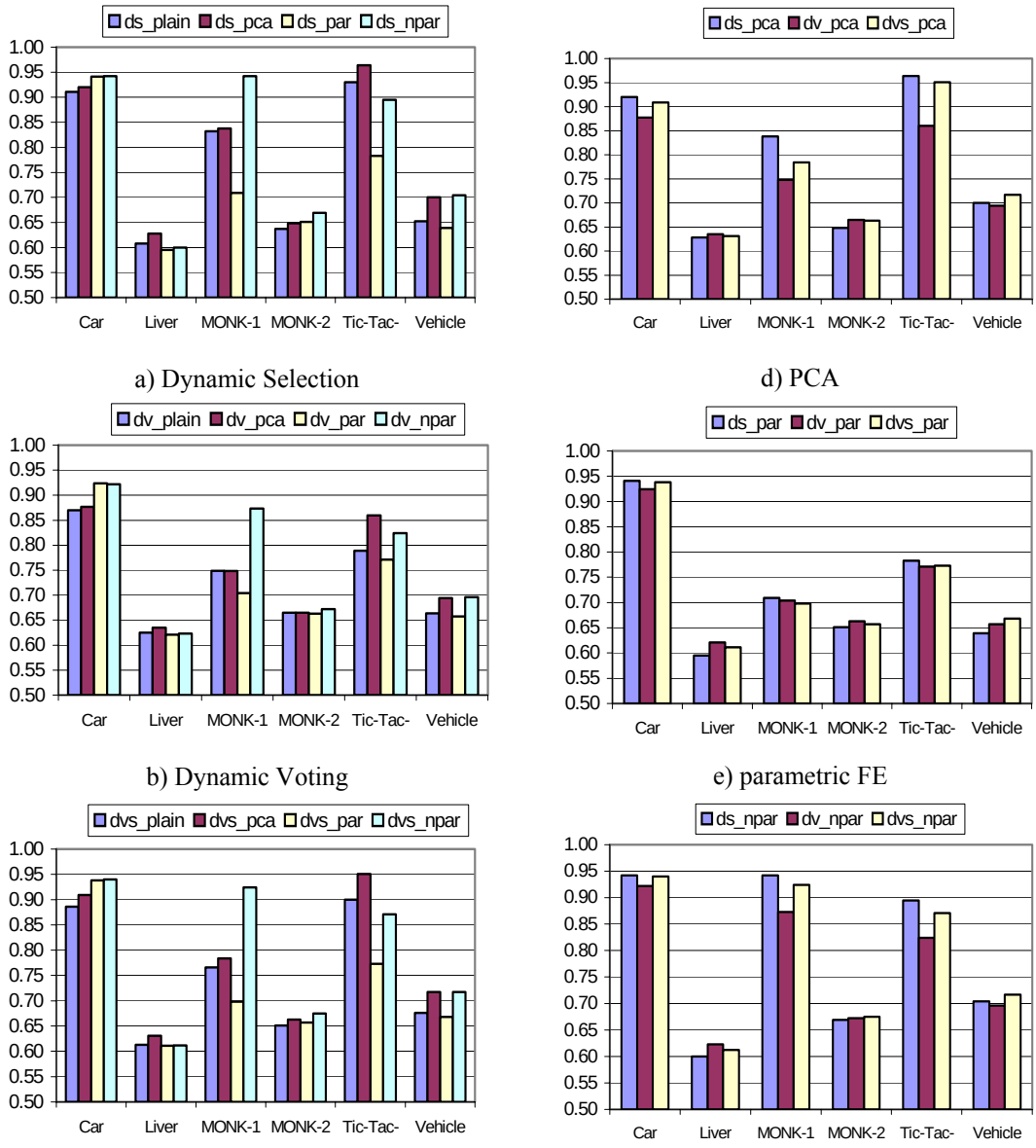

Figure 13 – Accuracy results on *Car, Liver, Monk-1, Monk-2, Tic-Tac-Toe* and *Vehicle* data sets grouped by integration method (a), b) and c)) and by feature extraction method (d), e), and f)).

Across the integration methods DS, DV and DVS (Figure 13, a-c) PCA is significantly better for two data sets than the other FE approaches (*Liver* and *Tic-Tac-Toe*), parametric FE is significantly worse for two data sets than PCA, nonparametric FE, and DS (*Monk-1* and *Tic-Tac-Toe*) and significantly worse for one data set than PCA and nonparametric FE (*Vehicle*). The nonparametric FE is significantly better for two data sets (*Monk-1* and *Monk-2*) than DS, and for one data set (*Monk-1*) than DV and DVS. Class-conditional FE approaches work well on the *Car* data set but work poorly (with DS) or slightly worse (with DV and DVS) on the *Liver* data set. However, we can see that patterns of the FE approaches behaviour are very similar with regard to any integration method.

For PCA (Figure 13, d) DV was the worst for three data sets (*Car, Monk-1*, and *Tic-Tac-Toe*) and it was never significantly better than DS and DVS. DS with PCA was the best on the Monk-1 data set and DVS with PCA was the best on the *Vehicle* data set. For parametric FE (Figure 13, e) DV was significantly better than DS on the *Liver* data set. However, there is no big difference in accuracy results with respect to the selected integration procedure. For nonparametric FE (Figure 13, f) DV was the worst on *the Car, Monk-1* and *Tic-Tac-Toe* data sets, and was slightly better than DS and DVS on the Liver data set. DS was the best on

Monk-1 and *Tic-Tac-Toe* data sets. DVS is slightly better than DS and DV on the *Vehicle* data set.

Since our main assumption about the benefit of applying DIC in the space of extracted features was the enhancement of neighborhood, we analysed how FE changes the neighborhood . We measured for each test instance, what proportion from the selected nearest neighbours belong to the same class as the true class value of the test instance. We calculated the proportions separately for 1, 3, 7, 15, 31, and 63 neighbours that is the same set that was used in the FEDIC experiments. The averages over the different number of neighbors are presented in Figure 13, and all the results are presented in Appendix B.

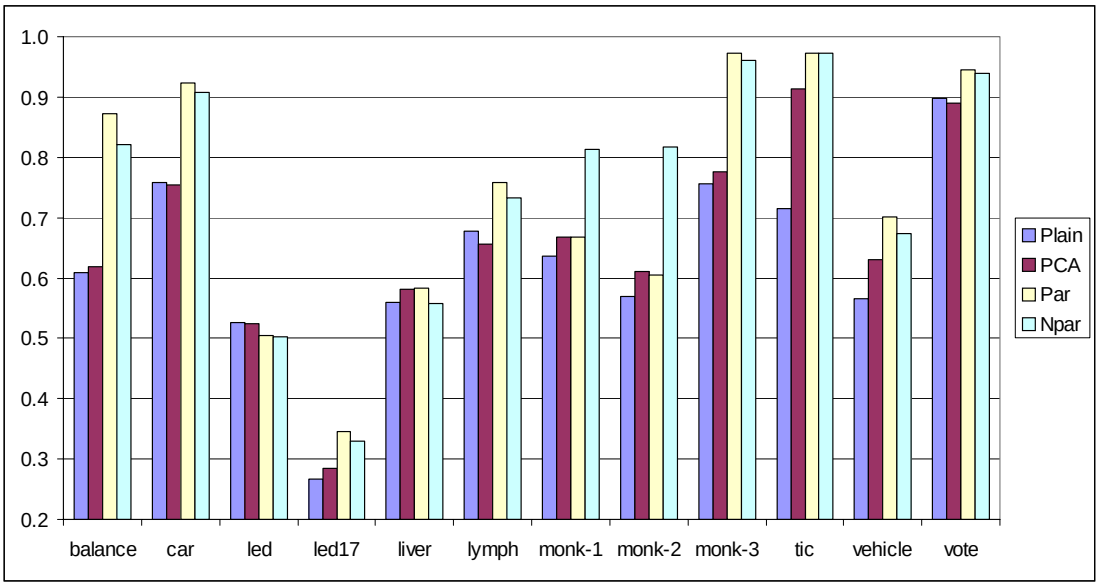

Figure 14 – Averaged over the number of test set instances, the proportion of the number of nearest neighbours with the same class label as a true class label of a new test instance to the total number of nearest neighbours.

We noticed that there exists complete correspondence of the results to the accuracy results in Table 4, irrespective of the number of nearest neighbors accounted by DIC. FE has no effect on the *Breast, Diabetes, Glass, Iris*, *Soybean, Thyroid,* and *Zoo* data sets. There is a small negative effect of FE on the *Led* data set. There is a strong positive effect of FE on the *Balance, Car, Led17, Monk-1, Monk-2, Monk-3, Tic-tac-toe,* and *Vehicle* data sets. The effect of FE on *Ionosphere, Liver, Lymph,* and *Vote* is not so strong, but still a significant effect for the output proportion of the instances belonging to the same class as a test instance. However, we can see that for FEDIC there is no effect of FE for the *Balance, Heart, Ionosphere, Led17, Monk-3* and *Voting* data sets. We suppose that these data sets are easy to learn for DIC and actually the same accuracy results are achieved by DIC in the space of the original features, because there is no room for further accuracy improvement.

Beside the accuracy results we also collected information about the threshold values that correspond to the highest accuracy and information about the number of features extracted on average (that correspond to the threshold values). In Table 6 we present the average threshold value and average number of features extracted for those 6 datasets where FEDIC outperforms DIC in the original space.

Table 6 – The average threshold value (thresh) and average number of features (feat) extracted for the 6 datasets where FEDIC outperformed DIC in the original feature space

	3-NN			PCA			PAR			NPar		
	PCA	Par	NPar	ds	dv	dvs	ds	dv	dvs	ds	dv	dvs
thresh	0.89	0.83	0.90	0.91	0.91	0.91	0.83	0.82	0.83	0.93	0.93	0.95
feat	11.8	1.7	4.7	11.7	11.7	11.7	1.7	1.5	1.7	6.4	6.4	6.7

The parametric approach extracted the least number of features and it was the least time-consuming approach. The nonparametric approach extracted more features due to its nonparametric nature, yet still was less time-consuming than PCA and classification in the space of original features. We should also point out that feature extraction speeds up dynamic integration in the same way as feature extraction speeds up a single classifier. This is as would be expected, since nearest-neighbour search for the prediction of local accuracies is the most time-consuming part of the application phase in dynamic integration, and it uses the same feature space as a single nearest-neighbour classifier does. Moreover, the two nearest-neighbour search processes (in dynamic integration and in a base classifier) are completely identical, and differ only in the number of nearest neighbours used to define the neighbourhood.

Threshold values and the number of extracted features are very similar in both base level classification (3NN) and DIC. However, we found that the nonparametric FE approaches extract more features for DIC compared to the single 3NN classifier and for dynamic voting with selection the most number of features is extracted on average.

6 Conclusion

Feature extraction as a dimensionality reduction technique helps to overcome the problems related to the "curse of dimensionality" with respect to the dynamic integration of classifiers. The experiments on 21 UCI data sets showed that the DIC approaches based on the plain feature sets had worse results in comparison to the results obtained using the FEDIC algorithm. This supports the fact that dimensionality reduction can often enhance a classification model.

The results showed that the proposed FEDIC algorithm outperforms the dynamic schemes on plain features usually only on those data sets, on which feature extraction for classification with a single classifier provides better results than classification on plain features. When we analyzed this dependency further, we came to a conclusion that feature extraction influences the accuracy of dynamic integration (in most cases) in a similar manner, as feature extraction influences the accuracy of a classifier.

The nonparametric approach and PCA were the best on average; the parametric approach was the worst. However, it is necessary to note that the parametric approach produces the most compact space of new features even if the resulting accuracy is not higher than either PCA or the nonparametric approach. It can therefore be argued that the parametric FE can still be beneficial from a performance point of view.

PCA for DIC showed stable results, as it also did with the 3NN classifier. Surprisingly, PCA was as good as the nonparametric approach for DIC, whilst its accuracy was significantly worse when compared to the nonparametric approach for the 3NN classifier.

We performed deeper analyses for those six data sets where FEDIC significantly outperformed dynamic integration in the space of original features.

Further research is needed to define the dependencies between the characteristics of a data set and the type and parameters of the feature extraction approach that best suits for it.

Some of the most important issues for future research raised by this work include how the algorithm could automatically determine whether FE is beneficial for a data set, what is the most suitable feature extraction method for the data set having certain characteristics, and what are the optimal parameter settings for the selected feature extraction method. Also of interest is, how the most appropriate dynamic integration scheme could be automatically identified, and what the optimal number of nearest neighbours in DIC is.

An improvement in accuracy results can be achieved if class-conditional FE techniques extract new features with respect to base classifier errors rather than with respect to initial class-label information of the instances (as it is in the train set).

Another interesting direction for further research is to consider whether FE should be applied globally (over the entire instance space) or locally (differing in different parts of the instance space). It can be seen that despite being globally high dimensional and sparse, data distributions in some domain areas are locally low dimensional and dense, e.g. in physical

movement systems (Vijayakumar & Schaal, 1997). We started to work in this direction in (Pechenizkiy *et al.*, 2006) analyzing how the natural clustering approach can facilitate local dimensionality reduction for supervised learning, and we plan to develop this approach further.

Acknowledgments:

This research is partly supported by COMAS Graduate School of the University of Jyväskylä, Finland and Science Foundation Ireland. We would like to thank the UCI ML repository of databases, domain theories and data generators for the data sets, and the MLC++ library for the source code used in this study. We would also like to thank anonymous reviewers for helpful comments and suggestions to the earlier version of this paper.

References

1. Aivazyan, S.A. Applied statistics: classification and dimension reduction. Finance and Statistics, Moscow (1989).
2. Aladjem, M. Parametric and nonparametric linear mappings of multidimensional data. Pattern Recognition 24(6) (1991), 543-553.
3. Bauer, E., Kohavi, R. An empirical comparison of voting classification algorithms: bagging, boosting, and variants. Machine Learning 36(1,2), (1999), 105-139.
4. Bellman, R., Adaptive Control Processes: A Guided Tour, Princeton University Press (1961).
5. Blake, C.L., Merz, C.J. UCI repository of machine learning databases. Dept. of Information and Computer Science, University of California, Irvine, CA (1998).
6. Brodley C., Lane T., Creating and exploiting coverage and diversity, in: Proc. AAAI-96 Workshop on Integrating Multiple Learned Models, Portland, OR, (1996) 8-14.
7. Cordella L., Foggia P., Sansone C., Tortorella F., Vento M. Reliability parameters to improve combination strategies in multi-expert systems, Pattern Analysis and Applications 2(3), (1999) 205-214.
8. Cost, S. & Salzberg, S. A weighted nearest neighbor algorithm for learning with symbolic features. Machine Learning 10(1), (1993) 57-78.
9. Dietterich T., Bakiri G. Solving multiclass learning problems via error-correcting output codes, Journal of Artificial Intelligence Research 2, (1995) 263-286.
10. Dietterich, T. Machine learning research: four current directions. AI Magazine 18(4) (1997) 97-136.
11. Fayyad U.M. Data Mining and Knowledge Discovery: Making Sense Out of Data, IEEE Expert 11(5), (1996), 20-25
12. Fukunaga, K. Introduction to statistical pattern recognition. Academic Press, London (1999).
13. Gama, J.M.P. Combining classification algorithms. Dept. of Computer Science, University of Porto, Portugal. PhD thesis (1999).
14. Giacinto, G. & Roli, F. Methods for dynamic classifier selection. In Proc. ICIAP '99, 10th International Conference on Image Analysis and Processing. IEEE CS Press, (1999) 659-664.
15. Hall, M.A. Correlation-based feature selection of discrete and numeric class machine learning. In Proc. Int. Conf. On Machine Learning (ICML-2000), Morgan Kaufmann, (2000) 359-366.
16. Hastie, T. & Tibshirani, R. Discriminant adaptive nearest-neighbor classification. IEEE Transactions on Pattern Analysis and Machine Intelligence 18 (6), (1996) 607-616.
17. Hao, H., Liu, C. & Sako, H. Comparison of Genetic Algorithm and Sequential Search Methods for Classifier Subset Selection, In Proc. 7th Int. Conference on Document Analysis and Recognition (ICDAR'03), Vol. 2, (2003) 765-769.
18. Heath D., Kasif S., Salzberg S. Committees of decision trees. In: B. Gorayska, J. Mey (eds.), Cognitive Technology: in Search of a Humane Interface, Elsevier Science, (1996) 305-317.
19. Jolliffe, I.T. Principal Component Analysis. Springer, New York, NY. (1986).
20. Kambhatla N., Leen T. K. Dimension reduction by local principal component analysis. Neural Computation 9, (1997) 1493-1516.
21. Kohavi, R. A study of cross-validation and bootstrap for accuracy estimation and model selection. In C.Mellish (ed.), Proc. 14th Int. Joint Conf. on Artificial Intelligence IJCAI-95. Morgan Kaufmann, San Francisco, CA (1995) 1137-1145.
22. Kohavi, R. Wrappers for performance enhancement and oblivious decision graphs. Dept. of Computer Science, Stanford University, Stanford, USA. PhD Thesis (1995).
23. Kohavi, R., Sommerfield, D., Dougherty, J. Data mining using MLC++: a machine learning library in C++. In Proc. 8th IEEE Conf. on Tools with AI. IEEE CS Press, (1996) 234-245.

24. Koppel M., Engelson S. Integrating multiple classifiers by finding their areas of expertise, in: AAAI-96 Workshop On Integrating Multiple Learning Models for Improving and Scaling Machine Learning Algorithms, Portland, OR, (1996) 53-58.
25. Krzanowski, W.J., & Marriott, F.H.C. Multivariate analysis part 2: Classification, Covariance structures and repeated measurements. London: Edward Arnold, (1994).
26. Liu H. Feature Extraction, Construction and Selection: A Data Mining Perspective, ISBN 0-7923-8196-3, Kluwer Academic Publishers (1998).
27. Merz, C.J. Dynamical selection of learning algorithms. In D.Fisher, H.-J.Lenz (eds.), Learning from data, artificial intelligence and statistics, Springer-Verlag, NY (1996).
28. Merz, C.J. Using correspondence analysis to combine classifiers. Machine Learning 36(1-2) (1999) 33-58.
29. Opitz D. Feature selection for ensembles. In: Proc. 16th National Conf. on Artificial Intelligence, AAAI Press, (1999) 379-384.
30. Opitz D., Shavlik J. 1996. Generating accurate and diverse members of a neural network ensemble, in: D. Touretzky, M. Mozer, M. Hassemo (eds.), Advances in Neural Information Processing Systems 8, MIT Press, 535-541.
31. Opitz, D. & Maclin, D. Popular ensemble methods: an empirical study. Journal of Artificial Intelligence Research 11 (1999), 169-198.
32. Oza, N.C., Tumer, K. Dimensionality reduction through classifier ensembles. Technical report NASA-ARC-IC-1999-124, Computational Sciences Division, NASA Ames Research Center, Moffett Field, CA (1999).
33. Pechenizkiy M., Tsymbal A., Puuronen S. Supervised Learning and Local Dimensionality Reduction within Natural Clusters: Biomedical Data Analysis, IEEE Transactions on Information Technology in Biomedicine, Special Post-conference Issue "Mining Biomedical Data", (2006).
34. Puuronen, S. & Tsymbal, A. Local feature selection with dynamic integration of classifiers. Fundamenta Informaticae 47(1-2), special issue “Intelligent Information Systems”, 91-117, (2001).
35. Puuronen, S., Terziyan, V., Tsymbal, A. A dynamic integration algorithm for an ensemble of classifiers. In Z.W. Ras, A. Skowron (eds.), Foundations of Intelligent Systems: ISMIS’99, Lecture Notes in AI, Vol. 1609, Springer-Verlag, Warsaw (1999) 592-6.
36. Quinlan J.R. Bagging, boosting, and C4.5, in: Proc. 13th National Conf. on Artificial Intelligence AAAI-96, Portland OR, AAAI Press, (1996) 725-730.
37. Quinlan, J.R. C4.5 programs for machine learning. San Mateo CA: Morgan Kaufmann, (1993).
38. Todorovski, L., Dzeroski, S. Combining multiple models with meta decision trees. In: Proc. Principles of Data Mining and Knowledge Discovery PKDD 2000, LNAI 1910, Springer, 54-64.
39. Tsymbal, A. Dynamic Integration of Data Mining Methods in Knowledge Discovery Systems, PhD Thesis, University of Jyväskylä, Finland, (2002).
40. Tsymbal, A., Puuronen S., Pechenizkiy M., Baumgarten M., Patterson D. Eigenvector-based feature extraction for classification. In Proc. 15th Int. FLAIRS Conference on Artificial Intelligence, Pensacola, FL, USA, AAAI Press (2002), 354-358.
41. Tsymbal, A., Puuronen S., Skrypnyk I. Ensemble feature selection with dynamic integration of classifiers. In Int. Congress on Comp. Intelligence Methods and Applications CIMA’2001, (2001).
42. Tsymbal, A., Puuronen, S., Patterson, D. Ensemble feature selection with the simple Bayesian classification. Information Fusion, Special Issue “Fusion of Multiple Classifiers”, Elsevier Science 4(2), (2003) 87-100.
43. Tsymbal, A., Pechenizkiy, M., Cunningham, P. Diversity in Search Strategies for Ensemble Feature Selection, Information Fusion 6(1), Special Issue "Diversity in Multiple Classifier Systems", Elsevier Science, 83-98, (2005).
44. Vijayakumar, S., Schaal, S. Local dimensionality reduction for locally weighted learning. In: Proc. of the IEEE Int. Symp. on Computational Intelligence in Robotics and Automation, (1997) 220-225.
45. Webb, G.I. MultiBoosting: a technique for combining boosting and wagging, Machine Learning 40(2), (2000) 159-196.
46. William D.R., Goldstein M. Multivariate Analysis. Methods and Applications. ISBN 0-471-08317-8, John Wiley & Sons (1984).
47. Wilson, D.R. & Martinez, T.R. Improved heterogeneous distance functions. Journal of Artificial Intelligence Research 6(1) (1997), 1-34
48. Wolpert, D. Stacked Generalization. Neural Networks, Vol. 5 (1992) 241-259.
49. Woods, K., Kegelmeyer, W., Bowyer, K. Combination of multiple classifiers using local accuracy estimates. IEEE Trans. on Pattern Analysis and Machine Intelligence 19(4), (1997) 405-410.
50. Wróblewski, J. Ensembles of classifiers based on approximate reducts, Fundamenta Informaticae 47(3,4), (2001) 351 – 360.

Appendix A – the basic accuracy results of the experiments on 21 UCI data sets

	DIC			FEDIC									Static		*Bayes*
	Plain			PCA			Par			NPar					
	ds	dv	dvs	ds	dv	dvs	ds	dv	dvs	ds	dv	dvs	ss	wv	
Balance	.896	.896	.898	.896	.895	.898	.896	.897	.896	.896	.896	.899	.896	.894	.896
Breast	.731	.744	.744	.730	.747	.747	.723	.731	.731	.731	.747	.747	.723	.744	.734
Car	.911	.870	.886	.920	.877	.909	.941	.924	.938	.942	.922	.940	.863	.854	.866
Diabetes	.758	.761	.763	.756	.762	.767	.749	.761	.755	.753	.761	.763	.761	.761	.753
Glass	.624	.679	.679	.622	.674	.674	.599	.603	.603	.610	.621	.621	.607	.623	.588
Heart	.820	.839	.836	.818	.839	.834	.806	.838	.835	.810	.839	.838	.826	.839	.845
Ionosphe	.915	.918	.918	.917	.920	.920	.915	.915	.915	.916	.917	.917	.915	.916	.908
Iris	.941	.938	.940	.933	.930	.927	.940	.929	.932	.935	.932	.933	.929	.919	.907
LED	.742	.744	.745	.742	.745	.746	.741	.744	.745	.741	.744	.745	.743	.751	.744
LED17	.640	.690	.683	.645	.690	.684	.642	.690	.684	.647	.690	.684	.643	.690	.637
Liver	.608	.625	.613	.628	.635	.631	.595	.621	.611	.600	.623	.612	.599	.615	.606
Lymphogr	.817	.830	.841	.812	.830	.840	.800	.828	.827	.811	.830	.836	.814	.824	.813
MONK-1	.832	.749	.766	.838	.748	.784	.709	.704	.698	.942	.873	.924	.746	.711	.743
MONK-2	.637	.665	.651	.648	.665	.663	.651	.663	.657	.669	.672	.675	.655	.664	.567
MONK-3	.984	.984	.984	.975	.972	.972	.985	.985	.985	.987	.986	.986	.97	.971	.969
Soybean	.991	1	1	.991	1	1	.991	1	1	.991	1	1	.991	1	1
Thyroid	.961	.956	.956	.958	.951	.951	.951	.946	.946	.955	.952	.952	.953	.943	.948
Tic-Tac-	.930	.789	.900	.964	.860	.951	.783	.771	.773	.895	.824	.871	.713	.730	.690
Vehicle	.652	.664	.676	.700	.694	.717	.639	.657	.668	.704	.696	.717	.582	.603	.594
Voting	.953	.925	.946	.953	.924	.946	.945	.924	.938	.949	.923	.943	.951	.922	.904
Zoo	.948	.960	.960	.948	.960	.961	.946	.959	.959	.947	.959	.961	.941	.948	.936
Average	***.823***	***.820***	***.828***	***.828***	***.825***	***.834***	***.807***	***.814***	***.814***	***.830***	***.829***	***.836***	***.801***	***.806***	***.793***

Appendix B – Averaged over number of test set instances proportion of the number of nearest neighbours with the same class label as a true class label of a new test instance to the total number of nearest neighbours.

	Plain						**PCA**						**Par**						**NPar**					
Data set	**1**	**3**	**7**	**15**	**31**	**63**	**1**	**3**	**7**	**15**	**31**	**63**	**1**	**3**	**7**	**15**	**31**	**63**	**1**	**3**	**7**	**15**	**31**	**63**
Balance	.600	.630	.649	.611	.590	.568	.650	.647	.644	.613	.585	.569	.880	.873	.873	.874	.871	.857	.890	.867	.841	.817	.779	.732
Breast	.740	.713	.711	.691	.684	.666	.710	.693	.676	.663	.649	.635	.670	.663	.659	.655	.653	.647	.660	.647	.639	.629	.619	.607
Car	.770	.813	.783	.751	.741	.685	.800	.797	.776	.754	.718	.683	.960	.950	.939	.921	.897	.876	.960	.947	.934	.909	.869	.824
Diabetes	.700	.683	.673	.665	.659	.650	.700	.687	.674	.665	.659	.650	.680	.687	.690	.689	.690	.686	.680	.680	.676	.675	.671	.665
Glass	.680	.620	.564	.502	.392	.338	.690	.617	.561	.505	.414	.339	.560	.533	.500	.465	.399	.339	.600	.550	.506	.463	.394	.335
Heart	.760	.757	.744	.733	.712	.665	.740	.747	.733	.712	.692	.650	.770	.773	.773	.775	.767	.743	.680	.683	.676	.668	.653	.621
Ionosphere	.860	.840	.813	.773	.716	.677	.880	.857	.833	.800	.744	.685	.820	.830	.830	.829	.825	.810	.840	.817	.807	.801	.787	.754
Iris Plants	.960	.943	.937	.899	.788	.475	.950	.940	.913	.893	.820	.476	.970	.963	.956	.954	.902	.476	.970	.963	.957	.955	.903	.476
LED	.650	.637	.631	.567	.418	.259	.640	.617	.626	.576	.423	.261	.620	.610	.610	.561	.387	.236	.620	.610	.611	.557	.386	.236
LED17	.350	.313	.283	.258	.222	.180	.390	.340	.304	.269	.225	.177	.440	.420	.389	.347	.282	.201	.440	.397	.367	.327	.263	.189
Liver	.620	.577	.557	.546	.534	.524	.630	.610	.586	.567	.554	.539	.600	.593	.586	.583	.577	.564	.570	.563	.559	.558	.555	.545
Lymph	.790	.743	.711	.671	.616	.533	.790	.727	.671	.636	.590	.523	.810	.797	.790	.788	.772	.597	.800	.787	.774	.755	.713	.573
MONK-1	.600	.707	.709	.633	.611	.558	.710	.717	.714	.667	.615	.588	.670	.667	.666	.669	.667	.667	.950	.897	.873	.807	.716	.642
MONK-2	.530	.570	.567	.593	.580	.577	.700	.640	.593	.581	.579	.571	.610	.610	.610	.611	.605	.590	.920	.900	.861	.812	.749	.663
MONK-3	.880	.817	.793	.734	.688	.630	.850	.857	.807	.772	.709	.661	.980	.980	.977	.971	.967	.963	.980	.980	.974	.963	.946	.922
Soybean	.990	.980	.861	.499	.245	.121	.980	.940	.756	.473	.251	.153	.960	.960	.867	.505	.245	.121	.970	.967	.867	.504	.249	.140
Thyroid	.960	.933	.906	.867	.797	.739	.930	.907	.876	.829	.764	.711	.940	.933	.921	.907	.824	.748	.950	.930	.911	.883	.805	.737
Tic	.670	.710	.823	.766	.685	.641	.970	.970	.966	.909	.859	.813	.970	.973	.973	.973	.974	.976	.970	.973	.973	.975	.974	.973
Vehicle	.680	.653	.606	.552	.487	.417	.750	.723	.683	.623	.543	.455	.730	.723	.714	.701	.680	.654	.760	.743	.714	.673	.607	.539
Voting	.920	.910	.907	.897	.885	.866	.920	.910	.899	.888	.876	.850	.940	.943	.944	.945	.947	.949	.940	.940	.943	.941	.940	.935
Zoo	.950	.910	.829	.686	.461	.231	.950	.900	.824	.677	.459	.231	.900	.860	.771	.636	.426	.237	.900	.860	.781	.636	.425	.237
Average	***.746***	***.736***	***.717***	***.662***	***.596***	***.524***	***.778***	***.754***	***.720***	***.670***	***.606***	***.534***	***.785***	***.778***	***.764***	***.731***	***.684***	***.616***	***.812***	***.795***	***.774***	***.729***	***.667***	***.588***