Dynamic Integration of Classifiers in the Space of Principal Components

Alexey Tsymbal[1], Mykola Pechenizkiy[2], Seppo Puuronen[2], David W. Patterson[3]

[1]Dept. of Computer Science, Trinity College Dublin, Dublin, Ireland
alexey.tsymbal@cs.tcd.ie
[2]Dept. of Computer Science and Information Systems, University of Jyväskylä, Jyväskylä, Finland
{mpechen, sepi}@cs.jyu.fi
[3]Northern Ireland Knowledge Engineering Laboratory, University of Ulster, U.K.
wd.patterson@ulst.ac.uk

Abstract. Recent research has shown the integration of multiple classifiers to be one of the most important directions in machine learning and data mining. It was shown that, for an ensemble to be successful, it should consist of accurate and diverse base classifiers. However, it is also important that the integration procedure in the ensemble should properly utilize the ensemble diversity. 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. 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 supervised 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. As a rule, FEDIC outperforms plain dynamic integration on data sets, on which both dynamic integration works (it outperforms static integration), and considered feature extraction techniques are able to successfully extract relevant features.

1 Introduction

Knowledge discovery in databases (KDD) is a combination of data warehousing, decision support, and data mining that indicates an innovative approach to information management. KDD is an emerging area that considers the process of finding previously unknown and potentially interesting patterns and relations in large databases [7]. Current electronic data repositories are growing quickly and contain huge amount of data from commercial, scientific, and other domain areas. The capabilities for collecting and storing all kinds of data totally exceed the abilities to

analyze, summarize, and extract knowledge from this data. Numerous data mining methods have recently 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 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 method selection is done dynamically taking into account characteristics of each new instance.

Recent research has proved the benefits of the use of ensembles of base classifiers for classification problems [6]. 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 up 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" [4] due to the drastic rise of computational complexity and classification error in high dimensions [9]. 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 last one are feature extraction and feature construction. The key difference between feature selection and feature transformation is that during the first process only a subset of original features is selected while the second approach is based on a generation of completely new features; feature construction implies discovering missing information about the relationships among features by inferring or creating additional features [14]. 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 [8].

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. We propose the FEDIC (Feature Extraction for Dynamic Integration of Classifiers) algorithm, which combines the dynamic selection and dynamic voting integration techniques (DS and DV) with the conventional Principal Component Analysis (PCA) and two supervised eigenvector-based approaches (that use the within- and between-class covariance matrices). The first eigenvector-based approach is parametric, and the other one is nonparametric. Both these take class information into account when extracting features in contrast to PCA [8, 10].

Our main hypothesis is that with data sets, where feature extraction improves classification accuracy when employing a single classifier (such as *kNN* or Naïve Bayes), it will also improve classification accuracy when a dynamic integration approach is employed. Conversely, with data sets, where feature extraction decreases (or has no effect) classification accuracy with the use of a single classifier, then feature extraction will also decrease (or will have no effect) classification accuracy when employing a dynamic integration approach.

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 nontransformed space.

2 Dynamic Integration of Classifiers

Recently the integration of classifiers has been under active research in machine learning, and different approaches have been considered [6]. The integration of an ensemble of classifiers has been shown to yield higher accuracy than the most accurate base classifier alone in different real-world problems. The two main approaches to integration are: first, the *combination approach*, where base classifiers produce their classifications and the final result is composed using those classifications, and second, the *selection approach*, where one of the classifiers is selected and the final result is the result produced by it.

The most popular and simplest method of combining classifiers is voting (also called majority voting and Select All Majority, SAM) [3]. In this simple method, the classification produced by a base classifier is considered as a vote for a particular class value, and the class value with the most votes is selected as the final classification. Weighted voting (WV) [3] and stacked generalization [25] are examples of more sophisticated combining methods.

One very popular but simple selection approach is CVM (Cross-Validation Majority) [12], which estimates the accuracy of each base classifier using cross-validation and selects a classifier with the highest accuracy.

CVM is an example of a *static* selection method that selects one base classifier for the whole data space. More sophisticated combining and selection methods use the estimates of the local accuracy of the base classifiers or meta-level classifiers, which predict the correctness of base classifiers for a new instance [15, 16]. These more sophisticated selection methods are called *dynamic* selection methods.

In [20] a dynamic approach that estimates the local accuracy of each base classifier by analyzing the accuracies of the base classifiers in near-by instances was elaborated. Instead of training a meta-level classifier that will derive the final classification using the classifications of the base classifiers as in stacked generalization, a meta-level classifier that will estimate the local errors of the base classifiers for each new instance and then use these errors to derive the final classification is trained. To predict the errors of base classifiers, the weighted nearest neighbor classification (WNN) is used [1].

The dynamic integration technique contains two main phases [19]. First, at the learning phase, the training set is partitioned into folds. The cross-validation technique is used to estimate the errors of base classifiers on the training set and a meta-level training set is formed. It contains all the attributes of the training instances

and the estimates of the errors of base classifiers on those instances. Second, at the application phase, a combining classifier is used to predict the performance of each base classifier for a new instance.

Two different functions implementing the application phase were considered in [19]: dynamic selection (DS) and dynamic voting (DV). At the application phase, DS selects a classifier with the least predicted classification error using the WNN procedure. DV uses the local errors as weights for the base classifiers. Then, weighted voting is used to produce the final classification.

3 Feature Extraction for Classification

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 [2, 8]. In other words, we are interested in finding a projection **w**:

$$\mathbf{y} = \mathbf{w}^T \mathbf{x}, \tag{1}$$

where **y** is a $p' \times 1$ transformed data point, **w** is a $p \times p'$ transformation matrix, and **x** is a $p \times 1$ original data point.

In [18] it was shown that the conventional PCA [10, 23] transforms the original set of features into a smaller subset of linear combinations that account for most of the variance of the original set. Although it is still probably the most popular feature extraction technique, it has a serious drawback, 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.

The usual approach to overcome this problem is to use some class separability criterion, e.g. the criteria defined in Fisher linear discriminant analysis, and based on the family of functions of scatter matrices:

$$J(\mathbf{w}) = \frac{\mathbf{w}^T \mathbf{S}_B \mathbf{w}}{\mathbf{w}^T \mathbf{S}_W \mathbf{w}}, \tag{2}$$

where $\mathbf{S}_B$ is the between-class covariance matrix that shows the scatter of the expected vectors around the mixture mean, and $\mathbf{S}_W$ is the within-class covariance, that shows the scatter of samples around their respective class expected vectors.

A number of other criteria were proposed in [8]. Both parametric and nonparametric approaches optimize the criterion (2) by the use of the *simultaneous diagonalization algorithm* [8]:

1. Transformation of **X** to **Y**: $\mathbf{Y} = \mathbf{\Lambda}^{-1/2} \mathbf{\Phi}^T \mathbf{X}$, 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 **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.

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 neighbor (kNN) technique is used for this purpose.

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. 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. 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. In [8] the parameter α was set to 1 and *nNN* to 3, but without any strict justification. In [20] it was shown that these parameters have different optimal values for each data set.

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 1, a scheme that illustrates the components of the FEDIC approach is presented. The FEDIC learning model consists of five phases: the training of the base classifiers phase, the feature extraction phase (FE), the dynamic integration phase (DIC), the model validation phase, and the model testing phase. The model is built using a wrapper approach [11] 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.

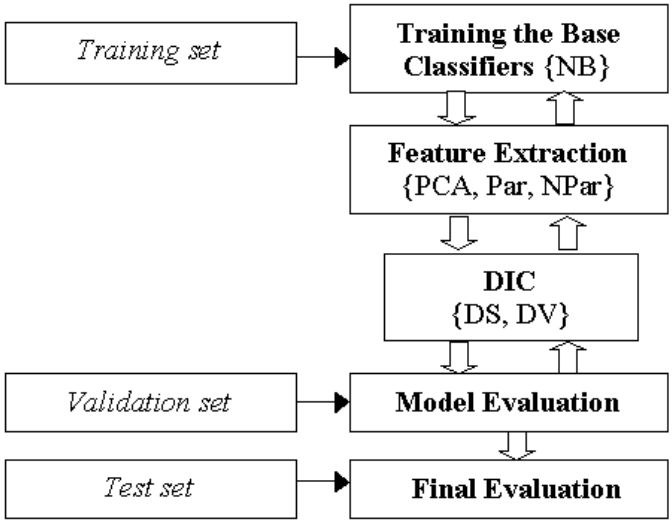

Fig. 1. Scheme of the FEDIC approach

In the next subsections we consider the training, feature extraction and dynamic integration of classifiers phases of the FEDIC algorithm.

4.2 The training of the base classifiers phase

The training phase begins with preprocessing which includes categorical features' binarization. Each categorical feature is replaced with a redundant set of binary features, each corresponding to a value of the original feature. An ensemble of classifiers is built from the pre-processed training data as shown in Figure 2. The training set **T** is partitioned into *v* folds. Then, cross-validation is used to estimate the errors of the base classifiers E_j ($\mathbf{x}^*$) on the training set and the meta-level training set $\mathbf{T}^*$ is formed. It contains the attributes of the training instances $\mathbf{x}_i$ and the estimates of the errors of the base classifiers on those instances $E_j(\mathbf{x}^*)$. The learning phase continues with training the base classifiers C_j on the whole training set.

$\mathbf{T}$ training set
$\mathbf{T}_i$ i-th fold of the training set
$\mathbf{T}^*$ meta-level training set
$c(\mathbf{x})$ classification of instance $\mathbf{x}$
$\mathbf{C}$ set of base classifiers
C_j j-th base classifier
$C_j(\mathbf{x})$ prediction of C_j on instance $\mathbf{x}$
$E_j(\mathbf{x})$ estimation of error of C_j on instance $\mathbf{x}$
$\mathbf{E}$ error matrix
m number of base classifiers

```
procedure training_phase(T,C)
      begin {fill in the meta-level training set T*}
        partition T into v folds
        loop for Ti ⊂ T, i = 1,...,v
          loop for j from 1 to m
            train(Cj,T-Ti)
          loop for x ∈ Ti
            loop for j from 1 to m
              compare Cj(x) with c(x) and derive Ej(x)
          collect (E1(x),...,Em(x)) into E
        T*=T|E
        loop for j from 1 to m
          train(Cj,T)
      end
```

Fig. 2. The training phase

4.3 The feature extraction phase

During this phase, feature extraction techniques are applied to the meta-level training set $\mathbf{T}^*$ to produce transformed meta-data with a reduced number of features. The pseudo-code of this process is shown in Figure 3. The formed meta-level data set is the input for the FE module where one of the three functions used: (1) *PCA_FE* that implements conventional PCA, (2) *Par_FE* that implements parametric feature extraction, or (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.

```
T*, T*'      meta-level and transformed meta-level data sets
S, Sb, Sw    total, between- and within-class covariance
              matrices
m            mean vector
Y            intermediate transformed space
Λ, Φ         eigenvalues and eigenvectors matrices
threshold    the amount of variance in the selected PCs
function PCA_FE(T*,threshold) returns T*'
begin
      S ← Σ_{i=1}^{n} (xi − m)(xi − m)^T
      Λ,Φ ← eigs(S) {the eigensystem decomposition}
      PPCA ← formPPCA(threshold, Λ, Φ)
            {forms the transformation matrix}
      return T*' ← PPCA T|E
end
```

function Par_FE($\mathbf{T}^*$,threshold) **returns** $\mathbf{T}^{*'}$
begin
$\mathbf{Y} \leftarrow$ 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 $\mathbf{Y}$ space}
$\mathbf{\Lambda}, \mathbf{\Phi} \leftarrow \text{eigs}(\mathbf{S}_B)$
$\mathbf{P}_{Par} \leftarrow \text{formP}_{\text{Par}}(threshold, \ \mathbf{\Lambda}, \ \mathbf{\Phi})$
return $\mathbf{T}^{*'} \leftarrow \mathbf{P}_{Par}\mathbf{Y} \mid \mathbf{E}$
end

function NPar_FE($\mathbf{T}^*$,threshold, kNN, alpha) **returns** $\mathbf{T}^{*'}$
begin
$\mathbf{Y} \leftarrow$ getYspace($\mathbf{T}^*$)
$w_{ik} \leftarrow \frac{\min_j \{d^{\alpha}(\mathbf{x}_k^{(i)}, \mathbf{x}_{nNN}^{(j)})\}}{\sum_{j=1}^{c} d^{\alpha}(\mathbf{x}_k^{(i)}, \mathbf{x}_{nNN}^{(j)})}$
$\mathbf{S}_B \leftarrow \sum_{i=1}^{c} n_i \sum_{k=1}^{n_i} w_{ik} \sum_{\substack{j=1 \\ j \neq i}}^{c} (\mathbf{x}_k^{(i)} - \mathbf{m}_{ik*}^{(j)})(\mathbf{x}_k^{(i)} - \mathbf{m}_{ik*}^{(j)})^T$
$\mathbf{\Lambda}, \mathbf{\Phi} \leftarrow \text{eigs}(\mathbf{S}_B)$
$\mathbf{P}_{NPar} \leftarrow \text{formP}_{\text{NPar}}(threshold, \ \mathbf{\Lambda}, \ \mathbf{\Phi})$
return $\mathbf{T}^{*'} \leftarrow \mathbf{P}_{NPar}\mathbf{Y} \mid \mathbf{E}$
end

function getYspace($\mathbf{T}^*$) returns $\mathbf{Y}$
begin
$\mathbf{S}_W \leftarrow \sum_{i=1}^{c} n_i \sum_{j=1}^{n_i} (\mathbf{x}_j^{(i)} - \mathbf{m}^{(i)})(\mathbf{x}_j^{(i)} - \mathbf{m}^{(i)})^T$
$\mathbf{\Lambda}, \mathbf{\Phi} \leftarrow \text{eigs}(\mathbf{S}_w)$
return $\mathbf{Y} \leftarrow \mathbf{\Lambda}^{-1/2}\mathbf{\Phi}^{\mathrm{T}}\mathbf{X}$
end

Fig. 3. The feature extraction phase

4.3 The dynamic integration phase

The transformed meta-data **T*'** is the input for the DIC module where the application phase of the dynamic integration process is performed - the combining classifier is used to predict the performance of each base classifier for a new instance. Either the function *DS_application_phase* or the function *DV_application_phase* (Figure 4) is

used for this purpose. Both functions begin with finding in the training set **T*'** a nearest neighborhood **NN** of the test instance **x** transformed with the corresponding transformation matrix calculated at the FE phase. The first function *DS_application_phase* implements Dynamic Selection. In the DS application phase the classification error E_j^* is predicted for each base classifier C_j using the WNN procedure and 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_application_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. For both functions the size of the set **NN** is an adjustable parameter.

---

x' an instance **x** transformed with P_{PCA}, P_{NPar} or P_{NPar}
W vector of weights for base classifiers
$\mathbf{T}^{*'}$ transformed meta-level training set
$E^*_j(\mathbf{x})$ prediction of error of C_j on instance **x**

function *DS_application_phase*(**T***', **C**, **x**) **returns** class of **x**
  begin
    NN=FindNeighborhood(**T***', **x'**, nn)
    loop for *j* **from** 1 **to** *m*

$$E_j^* \leftarrow \frac{1}{nn}\sum_{i=1}^{nn} W_{NN_i} \cdot E_j(\mathbf{x}_{NN_i})$$ {WNN estimation}

    $l \leftarrow \arg\min_j E_j^*$ {number of cl-er with min. E_j^*}
    {with the least global error in the case of ties}
    return $C_j(\mathbf{x})$
  end

function *DV_application_phase*(**T***', **C**, **x**) **returns** class of **x**
  begin
    NN=FindNeighborhood(**T***', **x'**, nn)
    loop for *j* **from** 1 **to** *m*

$$W_j \leftarrow 1 - \frac{1}{nn}\sum_{i=1}^{nn} W_{NN_i} \cdot E_j(\mathbf{x}_{NN_i})$$ {WNN estimation}

    return Weighted_Voting(**W**, $C_1(\mathbf{x}), \ldots, C_m(\mathbf{x})$)
  end

Fig. 4. The dynamic integration phase

We do not devote a separate subsection to the model validation and model evaluation phases since they are performed in a straightforward manner. 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 optimal parameters settings as based on the validation results. The selected model is then tested with a test data set.

In the next section we consider our experiments where we analyzed and compared the feature-extraction techniques described above.

5 Experimental Studies

We conducted the experiments on 20 data sets with different characteristics taken from the UCI machine learning repository [5]. The main characteristics of the 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, and the numbers of different types of features (categorical and numerical) included in the instances were presented in [21]. In [22] results of experiments with feature subset selection techniques with the dynamic selection of classifiers using these data sets were presented.

For each data set 70 test runs were made. In each test run a 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 [20]. Initial base classifiers were built using the Naïve Bayes on the training set and later refined using a hill-climbing cycle on the validation data set. The size of ensemble was selected to be equal to 25. It was shown that the biggest gain is achieved already with this number of base classifiers [2]. The diversity coefficient α was selected as it was recommended in [20] for each data set.

At each run of the algorithm, we collected accuracies for the four types of integration of the base classifiers: Static Selection (SS), Weighted Voting (WV), Dynamic Selection (DS), and Dynamic Voting (DV). 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) [24] was used for calculation of the distances in dynamic integration.

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 [13].

In Section 6 the results of dynamic integration with feature extraction using FEDIC are compared with the results when no feature extraction was used, and dynamic integration was therefore carried out in the space of original features.

6 Results and Discussions

The basic results of the experiments are presented in Table 1. The average classification accuracies of the 3-NN classifier (3-nearest neighbor search) are given for the three feature extraction techniques, namely PCA, parametric (Par) and nonparametric (NPar) approaches. Additionally, classification accuracy for the situation without feature extraction (Plain) is also shown. Then, in the same order, accuracies for the FEDIC approaches averaged over dynamic selection and dynamic voting schemes are presented. The last column contains classification accuracies for the static integration of classifiers (SIC) averaged over 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.

From Table 1 one can see that the nonparametric approach has the best accuracy on average both with the base classifier and with the dynamic integration scenarios. Although the parametric approach has extracted the least number of features, and it has been the least time-consuming approach, its performance has been unstable. The parametric approach has rather weak results on the Glass, Monk-1, and Tic data sets in comparison to the other feature extraction approaches. The scenario with parametric feature extraction has the worst average accuracy.

Table 1. Results of the experiments

Data set	3-NN Classifier				FEDIC				SIC
	PCA	Par	NPar	Plain	PCA	Par	Npar	Plain	Plain
Balance	.827	.893	.863	.834	.896	.897	.896	.896	.896
Breast	.721	.676	.676	.724	.747	.731	.747	.744	.744
Car	.824	.968	.964	.806	.920	.941	.942	.911	.863
Diabetes	.730	.725	.722	.730	.762	.761	.761	.761	.761
Glass	.659	.577	.598	.664	.674	.603	.621	.679	.623
Heart	.777	.806	.706	.790	.839	.838	.839	.839	.839
Ionospher	.872	.843	.844	.849	.920	.915	.917	.918	.916
Iris	.963	.980	.980	.955	.933	.940	.935	.941	.929
LED	.646	.630	.635	.667	.745	.744	.744	.744	.751
LED17	.395	.493	.467	.378	.690	.690	.690	.690	.690
Liver	.664	.612	.604	.616	.635	.621	.623	.625	.615
Lymph	.813	.832	.827	.814	.830	.828	.830	.830	.824
Monk-1	.767	.687	.952	.758	.838	.709	.942	.832	.746
Monk-2	.717	.654	.962	.504	.665	.663	.672	.665	.664
Monk-3	.939	.990	.990	.843	.975	.985	.987	.984	.971
Thyroid	.921	.942	.933	.938	.958	.951	.955	.961	.953
Tic	.971	.977	.984	.684	.964	.783	.895	.930	.730
Vehicle	.753	.752	.778	.694	.700	.657	.704	.664	.603
Voting	.923	.949	.946	.921	.953	.945	.949	.953	.951
Zoo	.937	.885	.888	.932	.960	.959	.959	.960	.948
Avg	.801	.803	.824	.766	.820	.805	.822	.815	.796

The nonparametric approach extracts more features due to its nonparametric nature, and still it was less time-consuming than the PCA and Plain classification. 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 one could expect, since nearest-neighbor 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-neighbor classifier does. Moreover, the two nearest-neighbor search processes (in dynamic integration and in a base classifier) are completely identical, and differ only in the number of nearest neighbors used to define the neighborhood.

The results of Table 1 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. For these data sets we have pairwise compared each FEDIC technique with the others and with static integration using the paired Student *t*-test with the 0.95 level of significance. Results of the comparison are given in Table 2. 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.

Table 2. Results of the paired *t*-test (win/tie/loss information) for data sets, on which FEDIC outperforms plain dynamic integration

	PCA	Parametric	Nonparam.	Plain	SIC
PCA		3/1/1	2/0/3	4/1/0	4/1/0
Parametric	1/1/3		0/2/3	2/2/1	3/1/1
Nonparam.	3/0/2	3/2/0		4/1/0	5/0/0
Plain	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	

There is an important trend in the results – 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, we will come to the conclusion 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.

The reason for that behavior is that both the meta-level learning process in dynamic integration, and the base learning process in base classifiers use the same feature space. Though, it is necessary to note, that the output values are still different in those learning tasks (these are local classification errors and the classes themselves correspondingly). Thus, the feature space is the same, and the output values to be predicted are different. This justifies that the influence of feature extraction on the accuracy of dynamic integration in comparison with the influence on the accuracy of a single classifier is still different to a certain degree.

In Figure 5 we summarize the accuracy results obtained on those data sets where at least some FEDIC-based technique (with PCA, parametric or nonparametric feature extraction) significantly outperforms the dynamic integration of classifiers on plain feature sets and the averaged results. It can be seen from the histogram that the nonparametric FEDIC shows the best accuracy on average of all the techniques considered. FEDIC and plain dynamic integration on average show almost the same results although we have to point out that this has happened due to the very unstable behavior of the parametric approach. The dynamic approaches significantly outperform the static ones.

7 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 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.

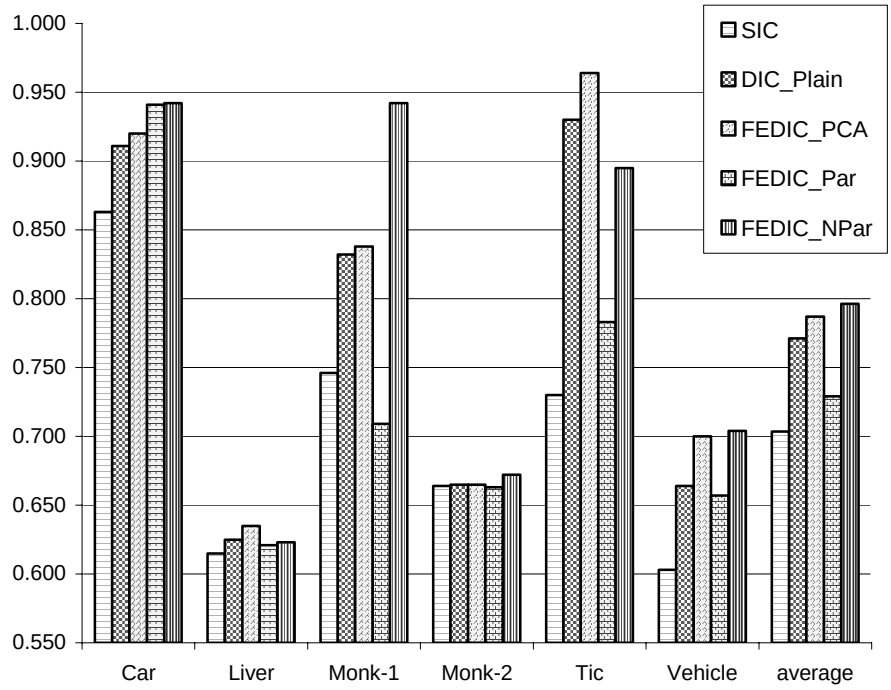

Fig. 5. Classification accuracy for data sets, on which the FEDIC algorithm outperforms plain dynamic integration

The results showed that the proposed FEDIC algorithm outperforms the dynamic schemes on plain features 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 influenced on the accuracy of dynamic integration in most cases in the same manner as feature extraction influenced on the accuracy of base classifiers.

The nonparametric approach was the best on average; however, it is necessary to note that each feature extraction technique was significantly better than all the other techniques at least on one data set. 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 it.

Some of the most important issues for future research to be raised by this work, include how the algorithm could automatically determine what transformation matrix should be chosen (i.e. what is the optimal feature extraction method) from the characteristics of the input data and what the optimal parameter settings for the selected feature extraction method should be. Also of interest is, how the most appropriate dynamic integration scheme could be automatically identified.

Acknowledgments: This research is partly supported by Science Foundation Ireland and COMAS Graduate School of the University of Jyväskylä, Finland. 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.

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, Vol.24(6) (1991), pp. 543-553.
3. Bauer, E., Kohavi, R. An empirical comparison of voting classification algorithms: bagging, boosting, and variants. Machine Learning, Vol. 36, Nos. 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 [http://www.ics.uci.edu/~mlearn/ MLRepository.html]. Dept. of Information and Computer Science, University of California, Irvine, CA (1998).
6. Dietterich, T.G. Machine learning research: four current directions. AI Magazine 18(4) (1997) 97-136.
7. Fayyad U.M. Data Mining and Knowledge Discovery: Making Sense Out of Data, IEEE Expert, Vol. 11, No. 5, Oct. (1996) pp. 20-25
8. Fukunaga, K. Introduction to statistical pattern recognition. Academic Press, London (1999).
9. Hall, M.A. Correlation-based feature selection of discrete and numeric class machine learning. In Proc. Int. Conf. On Machine Learning (ICML-2000), San Francisco, CA. Morgan Kaufmann, San Francisco, CA (2000) 359-366.
10. Jolliffe, I.T. Principal Component Analysis. Springer, New York, NY. (1986).

11. Kohavi, R. Wrappers for performance enhancement and oblivious decision graphs. Dept. of Computer Science, Stanford University, Stanford, USA. PhD Thesis (1995).
12. 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.
13. Kohavi, R., Sommerfield, D., Dougherty, J. Data mining using MLC++: a machine learning library in C++. In M.G.Radle (ed.) Proc. 8th IEEE Conf. on Tools with Artificial Intelligence. IEEE CS Press, Los Alamitos, CA (1996) 234-245.
14. Liu H. Feature Extraction, Construction and Selection: A Data Mining Perspective, ISBN 0-7923-8196-3, Kluwer Academic Publishers (1998).
15. 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).
16. Merz, C.J. Using correspondence analysis to combine classifiers. Machine Learning 36(1-2) (1999) 33-58.
17. Opitz, D. & Maclin, D. Popular ensemble methods: an empirical study. Journal of Artificial Intelligence Research 11 (1999), 169-198.
18. 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).
19. 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) 592-600.
20. Tsymbal, A., Puuronen, S., Patterson, D. Ensemble feature selection with the simple Bayesian classification. Information Fusion, Special Issue "Fusion of Multiple Classifiers", Elsevier Science (2003) (to appear).
21. 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.
22. Tsymbal A., Puuronen S., Skrypnyk I. Ensemble feature selection with dynamic integration of classifiers. In Int. ICSC Congress on Computational Intelligence Methods and Applications CIMA'2001, Bangor, Wales, U.K (2001).
23. William D.R., Goldstein M. Multivariate Analysis. Methods and Applications. ISBN 0-471-08317-8, John Wiley & Sons (1984), 587 p.
24. Wilson, D.R. & Martinez, T.R. Improved heterogeneous distance functions. Journal of Artificial Intelligence Research 6(1) (1997), 1-34
25. Wolpert, D. Stacked Generalization. Neural Networks, Vol. 5 (1992) 241-259.