Control and Cybernetics

vol. **29** (2000) No. 4

Similarity-based methods: a general framework for classification, approximation and association

by

Włodzisław Duch

Department of Computer Methods, Nicholas Copernicus University
ul. Grudziądzka 5, 87-100 Toruń, Poland
E-mail: duch@phys.uni.torun.pl

Abstract: Similarity-based methods (SBM) are a generalization of the minimal distance (MD) methods which form a basis of several machine learning and pattern recognition methods. Investigation of similarity leads to a fruitful framework in which many classification, approximation and association methods are accommodated. Probability $p(C|\mathbf{X}; M)$ of assigning class C to a vector $\mathbf{X}$, given a classification model M, depends on adaptive parameters and procedures used in construction of the model. Systematic overview of choices available for model building is presented and numerous improvements suggested. Similarity-Based Methods have natural neural-network type realizations. Such neural network models as the Radial Basis Functions (RBF) and the Multilayer Perceptrons (MLPs) are included in this framework as special cases. SBM may also include several different submodels and a procedure to combine their results. Many new versions of similarity-based methods are derived from this framework. A search in the space of all methods belonging to the SBM framework finds a particular combination of parameterizations and procedures that is most appropriate for a given data. No single classification method can beat this approach. Preliminary implementation of SBM elements tested on a real-world datasets gave very good results.

Keywords: similarity-based methods, kNN, optimization, feature selection, classification, approximation, associative memory.

1. Introduction

Recently, a general framework for similarity-based methods has been introduced (Duch, 1998). This framework is extended here, leading to new versions of similarity-based methods, including neural-like realizations. In pattern recogni-

of similarity-based methods, in statistics many clusterization methods belong to this group, in artificial intelligence the instance-based reasoning, memory-based reasoning or case-based reasoning methods (Mitchell, 1997) evaluate similarity to a set of prototype objects, and in neural networks many models are in fact variants of SBM. As a first step towards a general computational intelligence theory, integrating many learning methods within a single framework, various procedures and choices involved in creating similarity-based models are described here. These models operate on the same principle: given a set of objects create from them a set of reference objects $\{\mathbf{R}\}$ and introduce a similarity measure allowing to relate new query object $\mathbf{X}$ to the reference ones.

Four basic problems that such models may solve are: assign $\mathbf{X}$ to predetermined specific classes, map $\mathbf{X}$ to some numerical values, complete the missing features of $\mathbf{X}$, or create clusters that are in some respect homogenous. The first of these, supervised classification, has perhaps the widest applications, and therefore the outline of the SBM framework is presented from this perspective. Mapping problems – approximation and extrapolation – may be treated as classification with an infinite number of classes. Having selected a set of the most similar reference vectors to a given vector $\mathbf{X}$ a number of interpolation procedures may be applied to synthesize an approximate mapping. The same is true in the third case, completion of missing values. Known elements of the object $\mathbf{X}$ are used to find similar reference vectors and the missing parts are completed using approximation or classification procedures. SBM may thus serve as a basis for associative memories. Finally, clusterization or unsupervised classification problems require evaluation of similarity and thus also belong to the SBM. All of these methods may be useful in control problems.

A review of many approaches to classification and comparison of performance of 20 methods on 20 real world datasets has been done within the StatLog European Community project (Michie et al., 1994). More recently, the accuracy of 24 neural-based, pattern recognition and statistical classification systems has been compared on 11 large datasets by Rhower and Morciniec (1996). No consistent trends have been observed in the results of these large-scale studies. For each classifier one may find a real-world dataset, for which the results will be excellent, and another one for which the results will be quite bad. Therefore, in real world applications a good strategy is to find the best classifier that works for given data. Frequently, simple methods, such as the nearest neighbor methods or n-tuple methods (Rhower and Morciniec, 1996), are among the best. When selecting from the simplest classification models one should add different types of optimization parameters and procedures developing the model in the most promising direction in the space of all possible models belonging to the SBM framework.

Some of the best classification algorithms applicable to pattern recognition problems are based on the k-nearest neighbor (k-NN) rule (Krishnaiah and Kanal, 1982). Each training data vector is labeled by the class it belongs to

nearest to the unknown (query) vector $\mathbf{X}$ are found, and the class of vector $\mathbf{X}$ is determined by a 'majority rule'. The probability of assigning a vector $\mathbf{X}$ to a class C_i, $i = 1, \ldots, K$ is $p(C_i|\mathbf{X}; k) = N_i/k$, where N_i is the number of nearest vectors belonging to class C_i, $\sum_i^K N_i = k$. If $k = 1$ a single nearest neighbor determines the class of an unknown vector, i.e. $p(C_i|\mathbf{X}) = 0$ or 1. The asymptotic error rate of the k-NN classifier in the limit of large k and large number of reference vectors becomes equal to the optimal Bayesian values (Krishnaiah and Kanal, 1982). In real situations, the number of reference vectors is limited and small values of k may work better, therefore k should be optimized for each dataset.

Because the k-NN method is so simple it is frequently used as a standard reference for other classificators (surprisingly, very few computer programs for k-NN are around). Computational complexity of the actual classification is high, demanding for n reference vectors calculation of $\sim n^2/2$ distances and finding k smallest distances among them. Although Laaksonen and Oja (1996) claim that "For realistic pattern space dimensions, it is hard to find any variation of the rule that would be significantly lighter than the brute force method" various hierarchical schemes of partitioning the data space or hierarchical clusterization are quite effective in reducing the complexity of search from $O(n^2)$ to $O(n \log n)$. Even without any speedup of computations the datasets with several thousand of training patterns do not present any problems on modern personal computers. The search for the nearest neighbors is easily paralelizable and training time (selection of optimal k) is relatively short. Nearest neighbor methods are especially suitable for complex applications, where large training datasets are available. They are also used in the case-based expert systems as an alternative to the rule-based systems (see Waltz, 1995, for more than 200,000 reference patterns and millions of vectors for classification).

Only one neural model proposed so far is explicitly based on the nearest neighbor rule: the Hamming network (Lippmann, 1987, Floreen, 1991) computes the Hamming distances for the binary patterns and finds the maximum overlap (minimum distance) with the prototype vectors, realizing the 1-NN rule. Although other similarity-based methods presented here have natural neural-network type realizations we will concentrate more on presentation of the general framework rather than on the network implementation issues, since at this initial stage of the theoretical development implementation issues are of secondary importance. We will not spend much time on the actual methods of learning, based here on parameter optimization, neither. Other approaches to learning (Mitchell, 1997) may be useful in more complex situations. In the next section the general framework for SBM is presented and many novel elements outlined at each step of the classification process. The framework accommodates well-known classification methods and leads to new, unexplored methods. Examples of new methods and relations with known classification models, including some neural network models, are elucidated in the third section. Discussion and ref-

2. A framework for the similarity-based methods

Here, N is the number of features, K is the number of classes, vectors are in bold face while vector components are in italics.

The following steps may be distinguished in the supervised classification problem based on similarity estimations:

1) Given a set of objects (cases) $\{\mathbf{O}^p\}$, $p = 1,\ldots,n$ and their symbolic labels $C(\mathbf{O}^p)$, define useful numerical features $X_j^p = X_j(\mathbf{O}^p)$, $j = 1,\ldots,N$ characterizing these objects. This preprocessing step involves computing various characteristics of images, spatio-temporal patterns, replacing symbolic features by numerical values, etc.

2) Find a measure suitable for evaluation of similarity or dissimilarity of objects represented by vectors in the feature space, $D(\mathbf{X},\mathbf{Y})$.

3) Create a reference (or prototype) vectors $\mathbf{R}$ in the feature space using the similarity measure and the training set $\mathcal{T} = \{\mathbf{X}^p\}$ (a subset of all cases given for classification).

4) Define a function or a procedure to estimate the probability $p(C_i|\mathbf{X};M)$, $i = 1,\ldots,K$ of assigning vector $\mathbf{X}$ to class C_i. The set of reference vectors, similarity measure, the feature space, and procedures employed to compute probability define the classification model M.

5) Define a cost function $E[\mathcal{T};M]$ measuring the performance accuracy of the system on a training set $\mathcal{T}$ of vectors; a validation set $\mathcal{V}$ composed of cases that are not used directly to optimize model M may also be defined, and performance $E[\mathcal{V};M]$ measuring the generalization abilities of the model assessed.

6) Optimize the model M_a until the cost function $E[\mathcal{T};M_a]$ reaches minimum on the set $\mathcal{T}$ or on the validation set $E[\mathcal{V};M_a]$.

7) If the model produced so far is not sufficiently accurate add new procedures/parameters creating a more complex model M_{a+1}.

8) If a single model is not sufficient, create several local models $M_a^{(l)}$ and use an interpolation procedure to select the best model or combine results of a committee of models.

All these steps are mutually dependent and involve many choices described below in some details. The final classification model M is build by selecting a combination of all available elements and procedures. A general similarity-based classification model may include all or some of the following elements:

$M = \{\mathbf{X}(\mathbf{O}), \Delta(\cdot,\cdot), D(\cdot,\cdot), k, G(D), \{\mathbf{R}\}, \{p_i(R)\}, E[\cdot], K(\cdot), \mathcal{S}(\cdot)\}$, where:

$\mathbf{X}(\mathbf{O})$ is the mapping defining the feature space and selecting the relevant features;

$\Delta_j(X_j;Y_j)$ calculates similarity of features X_j, Y_j, $j = 1,\ldots,N$;

$D(\mathbf{X},\mathbf{Y}) = D(\{\Delta_j(X_j;Y_j)\})$ is a function that combines similarities of features to compute similarities of vectors; if the similarity function selected has metric properties, the SBM may be called the minimal distance (MD) method.

k is the number of reference vectors taken into account in the neighborhood

$G(D) = G(D(\mathbf{X}, \mathbf{R}))$ is the weighting function estimating contribution of the reference vector $\mathbf{R}$ to the classification probability of $\mathbf{X}$;
$\{\mathbf{R}\}$ is a set of reference vectors created from the set of training vectors $\mathcal{T} = \{\mathbf{X}^p\}$ by some selection and optimization procedure;
$p_i(\mathbf{R})$, $i = 1, \ldots, K$ is a set of class probabilities for each reference vector;
$E[\mathcal{T}; M]$ or $E[\mathcal{V}; M]$ is a total cost function that is minimized at the training stage; it may include a misclassification risk matrix $\mathcal{R}(C_i, C_j)$, $i, j = 1, \ldots, K$;
$K(\cdot)$ is a kernel function, scaling the influence of the error, for a given training example, on the total cost function;
$\mathcal{S}(\cdot)$ is a function (or a matrix) evaluating similarity (or more frequently dissimilarity) of the classes; if class labels are soft, or if they are given by a vector of probabilities $p_i(\mathbf{X})$, the classification task is in fact a mapping. The $\mathcal{S}(C_i, C_j)$ function allows to include a large number of classes, "softening" the labeling of objects that are given for classification.

Various choices of parameters and procedures in the context of network computations leads to a large number of similarity-based classification methods. Parameters of each model are optimized and a search is made in the space of all models M_a for the simplest and most accurate model that accounts for the data. Optimization should be done using validation sets (for example in cross-validation tests) to improve generalization. Starting from the simplest model, such as the nearest neighbor model, qualitatively new "optimization channel" is opened by adding the most promising new extension, a set of parameters or a procedure that leads to the greatest improvements. Once the new model is established and optimized, all extensions of the model are created and tested and a better model selected. The model may be more or less complex than the previous one (since feature selection or selection of reference vectors may simplify the model). The search in the space of all SBM models is stopped when no significant improvements are achieved by new extensions.

The steps involved in setting up a SBM model are presented below in a detailed way. Examples of well-known classification models and new models that result from the SBM framework are given in the next section.

2.1. Feature space and similarity of features

Frequently the database contains a numerical description of the objects and the preprocessing step involves only rescaling or standardization of the input data. Features used should allow to assign a new vector $\mathbf{X}$ to one of the classes with high reliability. The number of features created by the $\mathbf{X}(\mathbf{O})$ mapping should be as small as possible to avoid the "curse of dimensionality" (Bishop, 1995). In some cases a group of features of the same type may be aggregated and replaced by a single feature, using, for example, the linear combination $X_j = \sum_l s_{jl} X_l$. The s_{jl} scaling coefficients in this combination may be estimated in two ways. The first method is based on inexpensive local approach (Aha, 1998), which tries

for example the percentage of correctly classified training samples using only the X_j feature. The second method is global, treating s_{jl} as adaptive parameters that are optimized simultaneously using the total cost function $E[\mathcal{T}; M]$. In the multi-layer perceptron (MLP) network with two hidden layers the first layer should essentially perform aggregation and may sometimes be replaced by a linear layer. A more sophisticated approach, used in Support Vector Machines, is based on non-linear projection of feature vectors (Schölkopf et al., 1998).

In some methods the feature X_j taking the symbolic value $X_j = \tau_{k_j}$ is treated directly using an appropriate similarity function $\Delta_j(\tau_{k_j}, \tau_{l_j})$ that may be defined as follows. Define a characteristic class function: $\Gamma_m(\mathbf{X}) = 1$ if $\mathbf{X} \in C_m$, otherwise $\Gamma_m(\mathbf{X}) = 0$. The vector $\mathbf{X}$ with feature $X_j = \tau_{k_j}$ is denoted as $\mathbf{X}(X_j = \tau_{k_j})$. The number of vectors belonging to the class m with $X_j = \tau_k$ is $N_m(X_j = \tau_{k_j}) = \sum_X \Gamma_m(\mathbf{X}(X_j = \tau_{k_j}))$ and the total number of such vectors is $N(X_j = \tau_{k_j}) = \sum_m N_m(X_j = \tau_{k_j})$. The ratio of these two numbers estimates the probability $p(C_m|X_j = \tau_{k_j}) = N_m(X_j = \tau_{k_j})/N(X_j = \tau_{k_j})$ that given the symbolic value τ_{k_j} of feature X_j the whole vector belongs to the class C_m. Symbolic features that have similar probabilities should have high similarity:

$$\Delta_j(X_j = \tau_{k_j},\ Y_j = \tau_{l_j})^\alpha = \sum_m |p(C_m|X_j = \tau_{k_j}) - p(C_m|Y_j = \tau_{l_j})|^\alpha, \quad (1)$$

where α is an arbitrary exponent. The similarity of the two symbolic values of feature j is the highest (or dissimilarity is the lowest, $\Delta_j(\tau_{k_j}, \tau_{l_j}) = 0$) if both values τ_{k_j}, τ_{l_j} predict the same probabilities. The generalized Value-Difference Metric (VDM) for vectors with symbolic values is defined as:

$$D_{VDM}(\mathbf{X}, \mathbf{Y})^\alpha = \sum_j s_j \Delta_j(X_j = \tau_{k_j},\ Y_j = \tau_{l_j})^\alpha. \quad (2)$$

Since many classification methods require numerical inputs it is convenient to replace symbolic with numeric values. Replacing symbolic feature X_j with K-dimensional vector of probabilities $p(C_i|\mathbf{X}(X_j = \tau_{k_j}))$, $i = 1, \ldots, K$ allows to compute the same similarity values:

$$\Delta_j(\tau_{k_j}, \tau_{l_j})^\alpha = \sum_{m=1}^{K} |p(C_m|X_j = \tau_{k_j}) - p(C_m|Y_j = \tau_{l_j})|^\alpha. \quad (3)$$

Thus, $\Delta_j(\cdot, \cdot)$ is a Minkowski distance function in K-dimensional space. Note that since for two classes $p(C_1|\mathbf{X}) + p(C_2|\mathbf{X}) = 1$ only one probability $p(C_1|\mathbf{X})$ is sufficient to compute similarity:

$$\Delta_j(\tau_{k_j}, \tau_{l_j})^\alpha = 2|p(C_1|\mathbf{X}(X_j = \tau_{k_j})) - p(C_1|\mathbf{Y}(Y_j = \tau_{l_j}))|^\alpha. \quad (4)$$

The number of numerical features is the same as the number of symbolic features. For more than two classes ($K > 2$) the absolute value in the sum above

growth of the dimension of the feature space the Value-Difference Metric (2) should be used directly (Wilson and Martinez, 1997), or other methods that do not preserve probabilistic estimations of similarity may be used (Aha, 1998, Grąbczewski and Duch, 1999).

2.2. Similarity measures and feature scaling

Calculation of similarities is most often reduced to the Euclidean metric for continuous inputs and the Hamming metric for binary inputs. In a more general approach let us first define one-dimensional feature similarity functions $\Delta_j(X_j, Y_j)$, for example:

$$\Delta_j(X_j, Y_j) = X_j - Y_j \text{ a simple difference} \tag{5}$$

$$\Delta_j(X_j, Y_j) = |X_j - Y_j| \text{ an absolute value of the difference} \tag{6}$$

$$\Delta_j(X_j, Y_j) = \frac{X_j - Y_j}{max_j - min_j} \text{ renormalized difference} \tag{7}$$

$$\Delta_j(X_j, Y_j) = \frac{X_j - Y_j}{4\sigma_j} \text{ standardized difference} \tag{8}$$

$$\Delta_j(X_j, Y_j) = \delta(X_j, Y_j) \text{ overlap difference} \tag{9}$$

where in the last case Kronecker delta is used. Feature similarity may also be computed as the probabilistic value differences (2). Similarity is defined in this case via a data-dependent matrix with the number of rows equal to the number of classes and the number of columns equal to the number of features. Generalization for continuous values requires a set of probability density functions $p_{ij}(x)$, with $i = 1, \ldots, K$, $j = 1, \ldots, N$. This distance function may be used for symbolic values and combined with other distance functions for continuous attributes.

Generalized Minkowski metric involves two exponents, α and β, although frequently a single exponent $\alpha = \beta$ is used. Typical distance function compute:

$$D(\mathbf{X}, \mathbf{Y})^\beta = \sum_j^N \Delta_j(X_j, X_j)^\alpha \tag{10}$$

$$D(\mathbf{X}, \mathbf{Y}) = \max_j \Delta_j(X_j, Y_j) \text{ Maximum Value.} \tag{11}$$

Scaling factors weighting one-dimensional similarity functions allow to include different contributions of different attributes and are very useful global parameters. Minkowski distance with the scaling factors is defined as:

$$D(\mathbf{X}, \mathbf{Y}; s)^\beta = \sum_j^N s_j \Delta_j(X_j, Y_j)^\alpha;\ s_j \geq 0. \tag{12}$$

Euclidean metric corresponds to $\alpha = \beta = 2$, which is completely isotropic,

parallel to the axis than to the directions between the axis. In fact, the unit contour is a circle for Euclidean distance, a square with vertices in $(0, \pm 1)$ and $(\pm 1, 0)$ for Manhattan, approaching a square with vertices at $(\pm 1, \pm 1)$ for large $\alpha = \beta$, and a concave 4-arm star for $\alpha = \beta$ going to zero.

Methods of selecting optimal scaling factors for features were reviewed by Wettschereck et al. (1997a), where a five-dimensional framework to characterize different methods of scaling features has been proposed. Scaling is the simplest way of pre-processing the features. The scaling factors facilitate feature selection in an automatic way. Admitting only $s_j = 0, 1$ allows for simplified optimization of the scaling factors for feature selection.

Using the scalar product and the norm:

$$\langle \mathbf{X}|\mathbf{Y}\rangle = \sum_{j=1}^{N} X_j Y_j; \ \|\mathbf{X}\|^2 = \langle \mathbf{X}|\mathbf{X}\rangle \tag{13}$$

several other distance functions can be defined:

$$D_c(\mathbf{X}, \mathbf{Y}) = 1 - \frac{\langle \mathbf{X}|\mathbf{Y}\rangle}{\|\mathbf{X}\|\|\mathbf{Y}\|} \quad \text{Cosine distance} \tag{14}$$

$$D_d(\mathbf{X}, \mathbf{Y}) = 1 - \frac{2\langle \mathbf{X}|\mathbf{Y}\rangle}{\|\mathbf{X}\|^2 + \|\mathbf{Y}\|^2} \quad \text{Dice distance} \tag{15}$$

$$D_J(\mathbf{X}, \mathbf{Y}) = 1 - \frac{\langle \mathbf{X}|\mathbf{Y}\rangle}{\|\mathbf{X}\|^2 + \|\mathbf{Y}\|^2 - \langle \mathbf{X}|\mathbf{Y}\rangle} \quad \text{Jaccard distance} \tag{16}$$

$$D_C(\mathbf{X}, \mathbf{Y}) = \frac{\Delta_j(X_j, Y_j)}{\Delta_j(X_j, -Y_j)} \quad \text{generalized Canberra distance.} \tag{17}$$

Additional parameters that may be introduced in similarity measures are either global or local (different for each reference vector). In some applications (for example in psychology) similarities are not symmetric. The simplest extension to non-symmetric similarity function is obtained by introducing different scaling factors, depending on the sign of $X_j - Y_j$, for example:

$$D_n(\mathbf{X}, \mathbf{Y}; s)^\alpha = \sum_{j}^{N} (\max(0, s_{j+}(X_j - Y_j)) - \min(0, s_{j-}(X_j - Y_j)))^\alpha, \tag{18}$$

where two separate scaling factors $s_{j+}, s_{j-} \geq 0$ are used. This function provides $2N$ adaptive parameters. The Mahalanobis distance (Bishop, 1995) is obtained by applying a linear transformation to the input vectors. Alternatively, a metric tensor $G_{ij} = G_{ji}$ is introduced, providing $N(N+1)/2$ adaptive parameters:

$$D(\mathbf{X}, \mathbf{Y}; \mathbf{G})^2 = \sum^{N} G_{ij}(X_i - Y_i)(X_j - Y_j). \tag{19}$$

Any adaptive system may provide a distance function for similarity-based methods. For example, a typical MLP network may be trained on the differences of pairs of vectors $\{\mathbf{X} - \mathbf{Y}\}$, learning to predict the distance between the classes $\|C(\mathbf{X}) - C(\mathbf{Y})\|$. The output of the neural network is then used in k-NN or other similarity-based method (see Chiu and Kavanaugh, 1997, where a similar idea is pursued). A better way is to give an MLP both $\mathbf{X}$ and $\mathbf{X} - \mathbf{Y} = \{d^j(X_j) - d^j(Y_j)\}$ as input vectors, where $d^j(\cdot)$ is a set of the feature pre-processing functions (in the simplest case scaling factors). A non-symmetric similarity function $D(\mathbf{X} - \mathbf{Y}; \mathbf{X})$, smoothly changing between different regions of the input space, is obtained iteratively: for each training vector its k nearest neighbors are selected using initial similarity estimation, and after the first epoch of neural training the process is repeated using the new similarity function. Thus, MLP mappings may be used to create similarity functions most appropriate for a given data.

Minimization of in-class distances and maximization of between-class distances is used in some statistical methods (for example Fisher's discrimination). A distance function with such properties should be useful in similarity-based methods. A function of this sort is based on a combination of sigmoidal functions in each dimension:

$$d^j(X_j; \mathbf{p}) = d^j(X_j; \mathbf{a}_j, \mathbf{b}_j, \mathbf{c}_j) = \sum_{l=1}^{K_j} a_{jl}\sigma(b_{jl}X_j + c_{jl}), \tag{20}$$

where K_j determines the number of steps in the smoothed sigmoidal step function (Fig. 1). Using this transformation with the Minkowski metric a network of nodes computing such distances may be applied for classification or prediction,

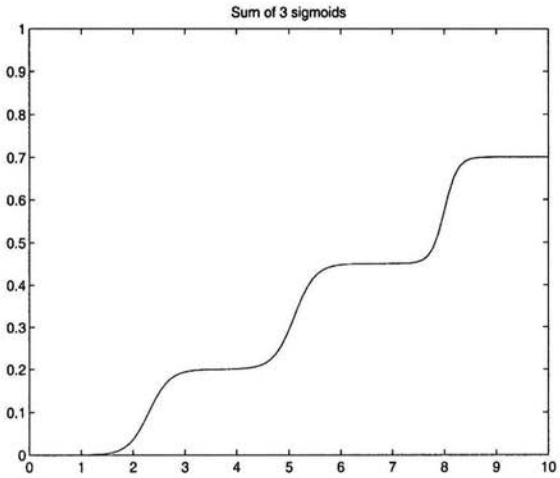

Figure 1. Sum of 3 sigmoidal functions provides a useful distance function al-

like any other neural network. It also could be used for extraction of logical rules from data, either fuzzy rules or – within the limits of high slopes – crisp logical rules. So far these ideas have not been tried in practice.

Calculation of distances may also be parameterized in a different way around each reference vector, providing a large number of adaptive parameters. Local coordinate systems with their origin placed at the reference vectors may provide either local scaling factors or local metric tensors. In specialized applications (for example in speech or handwritten letters recognition) invariant similarity measures are used – the "elastic matching" is defined by the shortest distance between two objects that are distorted in all possible ways while preserving their identity (class). Simard et al. (1993) introduced a simplification of this idea by measuring the distance between the tangent planes for the prototypes.

2.3. Feature selection

Scaling factors in the similarity function (12) allow for feature selection and feature scaling but since the global optimum of a cost function may be difficult to find (Duch and Grudziński, 1999b) simpler feature selection procedure may be useful. Many methods of feature selection and estimation of optimal scaling factors for features were reviewed by Wettschereck et al. (1997). These methods either iteratively optimize the scaling factors on the performance basis or assign fixed scaling factors by calculating mutual information between the values of features and the class of training samples or by summing probabilities (estimated through frequencies) of training vectors with non-zero values of features for a given class (per category feature importance). These scaling factors after binarization are used to select features.

Several simple feature selection procedures have been developed and tested specifically for the similarity-based methods (Duch and Grudziński, 1999b). In the feature-dropping algorithm features are removed consecutively, one at a time, and the best-first search (BFS) strategy is used. To achieve good generalization the leave-one-out test is performed on the training file and the change in accuracy is noted. Feature leading to the highest improvement of classification accuracy on the training file is selected as the least important and removed from the input set. If there is no improvement the feature that leads to a minimal degradation is selected. At each step all the remaining features are evaluated. At the end of the selection procedure all features are ranked according to their importance.

An approximate ranking of features is done at a lower cost. Assuming that features are independent and the effects of feature removal are additive only one test for each feature is done to determine the ranking. To make the method more robust features are ranked after averaging the results of crossvalidation tests with a single feature removed. An alternative is to perform the BFS feature-dropping algorithm using only a subset of features identified as promising during the approximate evaluation, for example using those features that

such as the beam-search, may be used if the number of features is not too large. After calculation of feature ranks crossvalidation tests with first M best features are performed for $M = 1, \ldots, N$. Usually, the best results are obtained with those features that on average were found useful (did not increase the accuracy after being dropped).

Search strategies may also be used for feature weighting. The cost function is simply the number of classification errors. Since features have real-valued weights, they have to be initially quantized, either with fixed precision or precision that is steadily increased during the progress of the search procedure. Non-gradient optimization methods that may be used for optimization of discontinuous cost function are expensive and may require a large number of function evaluations for convergence. Search methods for feature weighting are worth trying. Three such methods have been developed (Duch and Grudziński, 1999b): adding features starting from a single one, dropping features starting from all features and tuning the scaling factors, using the search procedure with systematic increase of the precision of the scaling factor's quantization.

Feature selection may be combined with regularization of the classification model. To lower the complexity of the model the cost function should include an additional penalty term, such as the sum of all s_j^2. Unrestricted optimization will of course lead to a very small values of all factors, therefore one should fix the scaling factor of the most important attribute at 1, optimizing over all other attributes (renormalization of the scaling factors is an alternative, but more complicated, solution). Features, for which the product of the scaling factors $s_j \max_{ik} |X_j^{(i)} - X_j^{(k)}|$ is small may be deleted without a significant loss of accuracy – after additional optimization of the scaling factors accuracy may even increase. In the framework of Wettschereck et al. (1997a) these methods use feedback, do not change feature representation, use global weights, do not use task specific knowledge and perform both feature selection and feature weighting.

Features may be selected globally, for all classification models or for all classes, leading to one set of features. For models that specialize in discrimination between pairs of classes or between a given class and all others, optimal features should be selected independently.

2.4. Missing values

The Value-Difference Metric treats the missing values like any other symbolic values, but if the missing feature is not symbolic it cannot be used directly. *Ad hoc* procedures based on replacing the missing values with class averages, the most frequent values, arbitrary constants, or ignoring these values, should be avoided. In statistics analysis of independent surveys in which some questions are not answered by some respondents and some questions are not asked in some surveys is known as the "multiple imputation" problem (Rubin, 1996). Assumptions about normal distributions used in the multiple imputation theory

Methods belonging to the SBM framework, such as the nearest neighbor method, may be used as associative memories in a natural way. Any part of the input vector $\mathbf{X} = (\mathbf{X}_d, \mathbf{X}_u)$ may be used to estimate the unknown input values $\mathbf{X}_d$ once the classification model is created. In the simplest case the undefined part $\mathbf{X}_u$ is predicted interpolating the values of nearest neighbors for the dominating class. Optimization of the model to increase classification accuracy in the $\mathbf{X}_d$ subspace should improve the results of $\mathbf{X}_u$ prediction.

An iterative technique of finding the missing values is recommended. In the first step a classification model M is created using the training vectors that do not contain missing features or using the largest subspace of features without missing values. This initial model is then used to calculate the probability of unknown values $\mathbf{X}_u$ by maximization of:

$$p(\mathbf{X}_u|\mathbf{X}_d; M) = \max_{u',i} p(C_i|(\mathbf{X}_{u'}, \mathbf{X}_d); M) \tag{21}$$

i.e. searching for the maximum of the probability given by the model M in the subspace of undefined features, with fixed point in the X_d subspace.

At a later stage, once all elements of the initial model are defined, feature selection and feature weighting procedures may be added. These procedures are closely connected with the definition of similarity measures.

2.5. Selection and weighting of reference vectors

SBM models may use all training data as the reference vectors. Reducing the size of the reference set leads to models of lower complexity, speeds-up classification and minimizes memory requirements (this is important not only in real-time applications – optimization of some parameters may require many evaluations of the cost function). It also helps to improve generalization capabilities of the classification system, especially for noisy data. Moreover, eliminating redundant cases and leaving only the most interesting prototypes may sometimes allow to understand the structure of the data, providing an alternative to the rule-based classifiers. Systems designed for on-line learning, where the number of the incoming vectors may in principle be infinite, must use partial memory (see Michalski, 1999), selecting the best prototypes.

Three groups of reference set selection algorithms may be distinguished: clusterization based algorithms, algorithms starting from the whole set, and algorithms starting from the empty set. K-means, dendrograms or other clusterization techniques may be used to select a relatively small number of initial reference vectors close to the centers of data clusters. Classification accuracy is checked on the remaining set and each wrongly classified vector is moved from the training to the reference set. Variants of this approach may use a validation set to determine best candidates for the reference set.

An alternative approach to selection of reference vectors that does not re-

that have all k nearest vectors from the same class are then removed from the reference set (k should be relatively large here, for example, $k = 10$). Removed vectors are far from cluster borders; all test vectors that fall in their neighborhood will be anyway unambiguously classified. This approach leads to a "hollow" cluster representation, leaving in the reference set only those vectors near the cluster borders. Variants of this approach may start with a large number of neighbors k' to remove vectors near the centers of clusters first, and decrease k' to the final k value in a few steps. Noisy data contain some training vectors that are surrounded by neighbors from different class; to remove them from the reference set the vectors that have all $k - 1$ neighbors from the same class and a single neighbor from another class should also be removed.

Another useful algorithm to select good reference vectors near class borders starts from the empty reference set. For every training vector $\mathbf{X}$ that belongs to the class $C(\mathbf{X})$ it finds k nearest vectors from classes $C_i \neq C(\mathbf{X})$. Those vectors are moved to the reference set. This algorithm also leaves in the reference set only vectors near the class borders.

In the SBL-PM (Similarity-Based Learner – Partial Memory) algorithm introduced recently (Grudziński and Duch, 2000) training vectors are sequentially removed and the prediction accuracy of the system on the whole training set is calculated after each removal. If the accuracy drops below a user-defined threshold, relative to the result of the leave-one-out test on the whole training set, the removed vector is placed in the reference set; otherwise it is eliminated. Unfortunately, because of the high computational costs this method may be used only for relatively small datasets or with classification models that have few adaptive parameters only, such as the k-NN method. More sophisticated methods, for example GIGA, using genetic algorithm for selection of the reference set (Fuchs, 1996), have even higher computational demands, but the results are not necessarily better.

In the on-line version of the method the system has to decide whether a new training case, coming from the input stream, should be added to the reference set (partial memory of past cases). An obvious approach, used in the IB2 procedure (Aha et al., 1991) is to check whether each new instance received is correctly classified using the reference set created so far and add it to this set only if it leads to an error. To make this algorithm more resistant to noise one may introduce a "candidate reference" vector, that is included only on the preliminary basis. Candidate reference vectors are then checked during subsequent learning: if they contribute to correct classification they are kept, but if their presence leads to errors they are removed.

Active selection of reference vectors may eliminate many training vectors from the reference set. Further optimization of their positions should decrease the training error. The reference vector $\mathbf{R}$ in the neighborhood of a training vector $\mathbf{X}$ should be updated as follows:

Here, η is the learning rate, slowly decreasing to zero during training, and Kronecker δ is 1 if the class $C(\mathbf{X}) = C(\mathbf{R})$ or 0 otherwise. Various rules for moving centers $\mathbf{R}$ are used: moving only the nearest neighbor, moving all k neighbors by the same amount, using distance-dependent η, decreasing η during training etc. (Laaksonen and Oja, 1996). One can also optimize a subset of vectors, for example only those that are close to the center of clusters.

Virtual Support Vectors (VSV) may be added to the reference set to improve classification rates. The simplest approach is to interpolate between existing training vectors and to add VSV between vectors of different classes that are near to each other. In cases when data clusters belonging to different classes are far from each other VSV help to shift decision borders between classes, improving generalization. If the clusters mix with each other or are very close VSV are not created at all because the vectors from different classes will be closer than a minimum threshold value.

Reference vectors $\mathbf{R}$ that are far from the query vector $\mathbf{X}$ should obviously have smaller contribution to the classification probability. Radial Basis Function (RBF) neural networks (Bishop, 1995) using Gaussian or inverse multiquadratic transfer functions are a particular example of the soft weighting minimal distance algorithms, where the number of prototypes included is not restricted, but the weighting function provides an effective cutoff. The conical radial function is favorite among fuzzy logic practitioners: zero outside the radius r and $1 - D(\mathbf{X}, \mathbf{R})/r$ inside this radius. Classification probability is calculated by the output node using the formula:

$$p(C_i|\mathbf{X}; r) = \frac{\sum_{m \in C_i} G(\mathbf{X}; \mathbf{R}^m, r)}{\sum_m G(\mathbf{X}; \mathbf{R}^m, r)};$$
$$G(\mathbf{X}; \mathbf{R}, r) = \max\left(0, 1 - \frac{D(\mathbf{X}, \mathbf{R})}{r}\right) \tag{23}$$

Here $G(\mathbf{X}; \mathbf{R}, r)$ is the weight estimating contribution of the reference vector $\mathbf{R}$ at some distance $D(\mathbf{X}, \mathbf{R})$. An almost constant weight value up to a distance r is provided by a sigmoidal function $\sigma(D(\mathbf{X}, \mathbf{R}) - r)$, falling to zero for larger distances (slope of the sigmoid may be used as an additional parameter here).

One may also use variable r equal to the distance to the k-th neighbor and the weighting function for the vectors inside this radius. If r_k is the distance to the k-th neighbor and $r_k \geq r_i$, $i = 1, \ldots, k-1$ then a conical weighting function

$$G(D) = 1 - D/\alpha r_k, \ \alpha > 1 \tag{24}$$

has values $G(0) = 1$ and $G(r_k) = 1 - 1/\alpha$. For large α the cone is very broad and all vectors receive the same attention; for α approaching 1 the furthest neighbor has weight approaching zero. Therefore, an SBM model with optimized α cannot be less accurate than the model that uses similarity to k prototypes

Wettschereck et al. (1997) propose the hyperbolic weighting scheme:

$$p(C|X;M) = \frac{\sum_{R\epsilon O_k(\mathbf{X})} \delta(C(\mathbf{X}), C) 1/(D(\mathbf{X},\mathbf{R}) + \epsilon)}{\sum_{\mathbf{R}\epsilon O_k(\mathbf{X})} 1/(D(\mathbf{X},\mathbf{R}) + \epsilon)} \tag{25}$$

where $O_k(\mathbf{X})$ is the neighborhood of $\mathbf{X}$ containing k reference vectors $\mathbf{R}$ and ϵ is a small constant used to avoid dividing by zero.

2.6. Estimation of classification probability

Classification models require a function or a procedure to estimate $p(C_i|\mathbf{X};M)$, probabilities of assigning vector $\mathbf{X}$ to class C_i. If the estimations do not sum to 1 they should be renormalized. Some methods may predict only the most likely class, in effect assigning probability 1 to this class and 0 to all others. In the k-NN method probabilities $p(C_i|\mathbf{X}) = N_i/k$, where $N_i \leq k$ is the number of neighbors belonging to the class C_i.

There is no guarantee that probabilities obtained from classifiers will give the accuracy of results above the base rate (majority rate). Classification models that are too complex frequently overfit the training data, especially if optimization of model parameters is done on the training set only. A simple way to correct these probabilities is to introduce an additional linear model. In the K-class problem the order of the classes is chosen in such a way that the majority class has the highest label. Probabilities $p(C_i|\mathbf{X};M)$ for $i = 1,\ldots,K-1$ are rescaled by parameters κ_i:

$$\begin{aligned} p_i(\mathbf{X}) &= \kappa_i p(C_i|\mathbf{X};M), \\ p_K(\mathbf{X}) &= 1 - \sum_{i=1}^{K-1} p_i(\mathbf{X}),\ \kappa_i \in [0,1] \end{aligned} \tag{26}$$

For $\kappa_i = 1$ nothing is changed and original probabilities are used; for $\kappa_i = 0$ all vectors are assigned to the majority class. Since this is a linear model it is easily optimized in the least-mean square sense. For two classes an explicit formula for the optimal $p_1(\mathbf{X}), p_2(\mathbf{X})$ may be written.

2.7. Cost function definition

Knowing the formula for classification probabilities a cost function $E[\mathcal{T};M]$ may be defined. It should include an estimation of the empirical risk of misclassification $\mathcal{R}(C_i,C_j)$, an estimation of the similarity (or dissimilarity) of the predicted classes $\mathcal{S}(C_i,C_j)$, a kernel function $K(\cdot)$, scaling the influence of the error on the total cost (for a given training example), and an appropriate regularization term to avoid overfitting. The empirical risk matrix $\mathcal{R}(C_i,C_j)$ measures the risk of assigning the class C_i when the true class is C_j. In most cases risk and

conceptually they are quite different. A high risk may be assigned to misclassification of two quite similar classes. In the simplest case $\mathcal{R}(C_i, C_j) = \delta_{ij}$ or $\mathcal{R}(C_i, C_j) = |i - j|$.

The simplest cost functions measure the number of classification errors, reported as the error (or accuracy) achieved on some dataset. The winning class $C_m(\mathbf{X})$, where $m = \arg\max_i p(C_i|\mathbf{X}; M)$, is compared with the true class $C(\mathbf{X})$ and the number of errors counted:

$$E(\mathcal{T}; M) = \sum_{\mathbf{X} \in \mathcal{T}} (1 - \delta(C_m(\mathbf{X}), C(\mathbf{X}))). \tag{27}$$

If classes are ordered in some meaningful way errors may be quantified and instead of the Kronecker delta the sum of the differences $(C_m(\mathbf{X}) - C(\mathbf{X}))^2$ or more generally of similarities $\mathcal{S}(C_m(\mathbf{X}), C(\mathbf{X}))$,

$$E(\mathcal{T}; M) = \sum_{\mathbf{X} \in \mathcal{T}} \mathcal{S}(C_m(\mathbf{X}), C(\mathbf{X})) \tag{28}$$

is minimized over all parameters and procedures involved in determination of $C(\mathbf{X})$. For $\mathcal{S}(C_i, C_j) = 1 - \delta_{ij}$ the cost function (27) is obtained, but a domain expert may provide specific similarity values.

Minimization of functions counting the number of classification errors is difficult because these functions are discontinuous. A "soft" evaluation of the cost

$$E(\mathcal{T}; M) = \sum_{\mathbf{X} \in \mathcal{T}} \sum_{i=1}^{K} \mathcal{R}(C_i, C(\mathbf{X})) \mathcal{S}\left(p(C_i|\mathbf{X}; M), p_i(\mathbf{X})\right) \tag{29}$$

allows to use inexpensive gradient methods to optimize parameters and procedures defining the model M. The soft dissimilarity function $\mathcal{S}(p(C_i|\mathbf{X}; M), p_i(\mathbf{X}))$ measures the difference between the predicted and the assumed probability p_i. Most often a quadratic function of the difference $(p(C_i|\mathbf{X}; M) - \delta(C_i, C(\mathbf{X})))^2$ is used. The error function becomes then the standard mean square error (MSE) function. Entropy-based and other cost functions are sometimes used as an alternative to MSE function (see Haykin, 1994). In general, the minimum of the MSE does not correspond to the minimum of the classification error. Only in the hard limit, when probabilities are zero or one, these two minima are identical.

A regularization term penalizing high complexity of the classification model may be added to the cost function. Regularization is quite effective in neural networks (Bishop, 1995), where it allows to enforce "smoothness" on the mappings performed by a system based on a large number of homogenous parameters (weights). Regularization lowers the number of effective parameters of the model and prevents overfitting of the data, thus improving the general-

performance on the validation set, rather than directly on the training set, is a form of regularization. Noise added in a controlled way to the data may regularize the model (Bishop, 1995). A bootstraping technique used in statistics (Breiman, 1998) is also an effective regularization technique.

Kernel function plays a different role than dissimilarity or risk functions. A kernel function $K(D)$, for example a Gaussian function $K(D) = e^{-D^2/2\sigma^2}$, measures the influence of the reference vectors on the total error. $D = D(\mathbf{X}, \mathbf{R})$ measures here the distance of the vector $\mathbf{X}$ to the nearest reference vector $\mathbf{R}$ or to a set of all reference vectors $\mathbf{R}^{ref}$. In local regression based on the minimal distance approaches (Atkenson et al., 1997) the error function is simply

$$E(\mathcal{T}; M) = \sum_m K(D(\mathbf{X}^m, \mathbf{R}^{ref}))(F(\mathbf{X}^m; M) - y^m)^2 \tag{30}$$

where y^m are the desired values for $\mathbf{X}^m$ and $F(\mathbf{X}^m; M)$ are the values predicted by the model M. If $K(D)$ has a sharp high peak around $D = 0$ the function $F(\mathbf{X}; M)$ will fit the values corresponding to the reference input vectors almost exactly but will admit larger errors for other values. This may be regulated by changing the dispersion σ of the Gaussian kernel function. This is not the same as the weighting function $G(D)$ which is used to estimate the influence of distance on contribution to classification probability. In classification problems kernel function will determine the size of the neighborhood around the known cases in which accurate classification is required.

2.8. Optimization and additional parameters/procedures

Optimization method that should be used to minimize the cost function $E[\mathcal{T}; M]$ depends on the type of model used. To improve generalization a validation set $\mathcal{V}$ may be used, composed of vectors that are not in the training set and not in the test set. To avoid overfitting of the model to the training data the $E(\mathcal{V}; M)$ cost function should be minimized instead of the $E(\mathcal{T}; M)$. The reference vectors for the model M are selected using the training set $\mathcal{T}$ only, but features are selected and parameters are optimized to minimize $E(\mathcal{V}; M)$. For example, the leave-one-out error is minimized when the sum runs over all training examples $\mathbf{X} \in \mathcal{T}$ except for one vector $\mathbf{X}^p$. The model M does not contain this $\mathbf{X}^p$ vector in the reference set while $p(C_i|\mathbf{X}^p)$ is computed. The averaged error for all $p = 1, \ldots, n$ should be minimized – this is quite simple in the k-NN method, where the only parameter optimized (k) has integer values.

For real-valued parameters multistart gradient methods seem to be the most effective in optimization, if formulas for gradients of the error function can be derived. Some models may be efficiently optimized by organizing gradient computations in a neural network-like style. Real-valued parameters are provided by transformation of features $d(X_j)$, similarity measures of features $\Delta(\cdot)$ and

estimating contribution of the reference vector $\mathbf{R}$ to the classification probability $G(D) = G(D(\mathbf{X}, \mathbf{R}))$.

In some applications the training vectors may be mislabeled. This effect may be included by assigning probabilities of classes $p_i(\mathbf{R})$, rather than class labels (equivalent to binary probabilities), to the training vectors. Probabilites assigned to the reference vectors allow for soft-weighting of the class labels. An interesting possibility is to treat these probabilities as adaptive parameters. This should allow the classifier to reach base rate errors in regions where a few outliers exist. A simple method to adapt these probabilities is to start from the initial labels, i.e. class probability $p(C_i|R) = \delta(C_i, C(R))$ and modify it, so as to account for the neighborhood, adding just one parameter to preserve normalization:

$$p(C_i|R) \leftarrow (1-\gamma)p(C_i|R) + \gamma p(C_i|R; M) \tag{31}$$

i.e. *a priori* probabilities are corrected by the data. The γ parameter should now be optimized. More complex models with several parameters may of course be considered. Optimization of class probabilities is a form of data regularization, leading to models that are more resistant to noise in the data.

2.9. Ensemble of models

An adaptive system may include several models M_l and an interpolation procedure to select between different models or average results of a committee of models. Such averaging with boosting procedures for selection of training vectors leads to creation of stable and accurate classifiers (Breiman, 1998). Simple averaging, or linear combination of several models is most frequently used:

$$P(C_i|\mathbf{X}; M) = \sum_{l=1}^{N} W_l \; p(C_i|\mathbf{X}; M_l). \tag{32}$$

Least square minimization (LSM) procedure is used to determine W_l coefficients. When creating ensembles one should use all the information available. Since we know for which training vectors $\mathbf{R}^m$ each model makes an error it seems reasonable to use this information in making an ensemble. Coefficients of linear combination should depend on the distance between $\mathbf{X}$ and those regions around reference vectors $\mathbf{R}_l^m$ of the feature space where model M_l works poorly, therefore:

$$P(C_i|\mathbf{X}; M) = \sum_{l=1}^{N} \sum_{k} W_l D(\mathbf{X}, \mathbf{R}_l^m) p(C_i|\mathbf{X}; M_l) \tag{33}$$

should be a good choice. Identical LMS optimization is used as in the previous case. Probabilities are obtained after renormalization:

$$p(C_i|X; M) \leftarrow P(C_i|X; M) / \sum P(C_i|X; M). \tag{34}$$

Instead of a single model that tries to provide one distance function in the whole input space, several local distance functions may be defined around the main prototypes obtained using some initial clusterization method. This corresponds to a local coordinate systems that may have quite different optimal scaling factors and orientations.

Various procedures for combining results of different models may be defined, the simplest based on the selection of the submodel with the minimum distance from the vector given for classification, and the more sophisticated based on the estimation of confidence of each submodel in a given region of the input space. Using more than one model provides more adaptive parameters and should improve the results. New submodels may also be introduced in an incremental fashion, adding local systems in the regions of space where classification is less accurate.

3. Examples of SBM models

Many pattern recognition, machine learning and neural network models may be accommodated in the SBM framework. One way to use this framework is to start with the simplest model and develop it in the most promising direction by adding new optimization parameters and procedures. For example, starting from the simplest k-nearest neighbor method with Euclidean distance measure on standardized data one may consider the following improvements: optimization of the number of neighbors, optimization of the distance function, sophisticated distance functions (such as in Fig. 1), soft weighting, selection of features, selection and optimization of reference vectors, using several models and many other options.

Each step towards more complex model decreases the bias of the classifier, but may increase its variance (Breiman, 1998), therefore after each step the model should be validated and only if the greater complexity is justified by higher accuracy more complex models should be accepted, otherwise a different type of optimization should be used.

A few examples of known and novel methods belonging to the SBM framework are given below.

3.1. k-NN model

In the k-NN model $p(C_i|\mathbf{X};M)$ is parameterized by $p(C_i|\mathbf{X};k,D(\cdot),\{\mathbf{X}\})$, i.e. the whole training dataset is used as the reference set, k nearest prototypes are included with the same weight, and a typical distance function, such as the Euclidean or the Manhattan distance, is used. Probabilities are calculated as the ratio of the number of neighboring vectors belonging to the class C_i to the number of all neighbors included, $p(C_i|\mathbf{X};M) = N_i/k$, and the most probable

The restriction to k neighbors is realized by a hard-sphere metric distance function $D(\mathbf{X}, \mathbf{X}^m)$ with radius such that exactly k neighboring vectors $\mathbf{X}^m$ fall inside it. The type of the distance function $D(\cdot)$ and k are usually the only parameters optimized in the k-NN model. For $k = 1$ there is no error on the training set, but already for $k = 2$ the training vector near the class border may have the nearest vectors from two different classes. Therefore the error on the training set, equal to zero for $k = 1$, grows for $k > 1$ but may decrease for larger values of k. The leave-one-out test is recommended to optimize k using the training set data only. This type of test is particularly easy to perform in the k-NN method since there is no learning phase, unless the metric function is parameterized. For two-class problems odd k values are recommended to avoid ties that arise when the same number of neighbors from different classes is found. For the K-class problem $k = 1, K+1, 2K+1, \ldots$ avoids the ties but is a severe restriction on the choice of k. Ties may be resolved either by: a) rejecting cases in which tie occur; b) adding one or more extra neighboring vectors until the tie is broken; c) decreasing the number of neighboring vectors; d) randomly breaking the tie; e) selecting class with the largest *a priori* probability; f) leaving probabilities instead of yes-no decisions; g) using Eq. (26) to correct the computed probabilities.

Details of the k-NN procedure are rarely given in papers on applications and it is not always clear how ties are broken. In our experience the last two options are the most appropriate. Adding more vectors to break the tie seems to be reasonable, although in real applications differences in classification accuracy are sometimes negligible since ties do not occur if real-valued features are used.

The simplest error function used in optimization of k and the selection of the type of similarity function $D(\cdot)$ is:

$$E(\mathbf{X}; k, D) = \sum_{p=1}^{K} (1 - \delta(C(\mathbf{X}^p), C_j(\mathbf{X}^p))$$
$$C_j(\mathbf{X}^p) \leftarrow \max_j \; p(C_j|\mathbf{X}^p; M) \tag{35}$$

where $C(\mathbf{X}^p)$ is the true class of the vector $\mathbf{X}^p$ while $C_j(\mathbf{X}^p)$ corresponds to the best k-NN recommendation. This function should be minimized in respect to all adaptive parameters of the model M (here only k and the type of D function). In problems where a natural similarity of classes is defined or a risk function has been given cost functions (29) and (28) should be used.

3.2. r-NN models

Instead of enforcing exactly k neighbors the radius r may be used as an adaptive parameter. The number of classification errors, or the probability of classification $p(C_i|\mathbf{X}; r) = N_i / \sum_l N_l$, is then optimized using the leave-one-out method

network realization of this algorithm. r-NN may reject some vectors $\mathbf{X}$ if no reference vectors fall into the r radius of $\mathbf{X}$ or if equal probability of classification for several classes is obtained, but one could also consider a method with variable r (increased until a unique classification is done) to avoid such problems.

Introduction of variable radii r_i for each reference vector instead of one universal radius in the input space improves the method by further increasing the number of adaptive parameters significantly. Development along this line leads to the Restricted Coulomb Energy (RCE) classifier introduced by Reilly, Cooper and Elbaum (1982), which may be treated as the hard limit approximation of the Gaussian-based RBF network. If no neighbors are found around the training vector $\mathbf{X}$, the new spheres (reference vectors) are added with largest radius such that the sphere does not overlap with the spheres of other classes. If the new training vector falls into the range of a sphere of a wrong class, the radius of this sphere is shrinked to leave the vector outside of the sphere. Positions of the spheres are not optimized in the RCE algorithm – this would lead in the direction of LVQ algorithms (Laaksonen and Oja, 1996) – but voting methods for the committees of classifiers were used with success (Wasserman, 1993).

The number of radiuses r_i may be reduced by using only a few independent values in selected input space areas. One could also optimize components of one radius (i.e. not just a total distance but separate distances for individual input features), but this would give the same result as optimization of the metric function described below. To reduce the number of parameters, variable radiuses should be attached only to the centers of clusters. To assure smooth transition between different regions of the input space interpolation of the r values from the nearest cluster centers is recommended.

Although r-NN model is quite simple it does not seem to be used and little is known about it. Our preliminary test showed that on same datasets it gives better results than k-NN. A combination of these two nearest neighbor methods could also be considered using Eq. (32) or (33).

3.3. Soft weighting k-NN and r-NN methods

A natural generalization of the r-NN method is obtained by introducing the $G(D)$ weighting function instead of sharply cutting off the neighbors taken into account at the specified radius r. The Gaussian classifier (see Wasserman, 1993, Krishnaiah and Kanal, 1982) also belongs to this category. In the simplest version of the RBF algorithm Gaussian functions are used and only one parameter – dispersion – is optimized (Bishop, 1995). Independent optimization of all N components of the dispersion vector has the same effect as optimization of the scaling factors s_j in the soft-weighted NN-r method.

Other methods of weighting discussed in the previous section may be tried with the k-NN or r-NN method. The effect of weighting is more pronounced

and α is optimized in the $G(D) = 1 - D/\alpha r_k$ function, the results should be close to the r-NN method, but if both k and α are optimized the results should be better.

3.4. RBF, FSM, LVQ and fuzzy systems

In RBF networks Euclidean distance functions $D(\mathbf{X}, \mathbf{R}^j) = \|\mathbf{X} - \mathbf{R}^j\|$ are assumed, and radial, for example Gaussian $G(D) = \exp(-D^2)$, weighting functions are used. Essentially, RBF is a minimal distance soft weighted method with no restrictions on the number of neighbors – reference vectors $\mathbf{R}^j$ that are near influence the probabilities of classification more than those that are far. The SBM framework suggests that there is nothing special about this choice of distance function and the weighting function (see the conical radial weighting function (23) and other possibilities of weighting).

Optimization of the positions of reference vectors $\mathbf{R}^m$ leads to the Learning Vector Quantization type of methods (LVQ, Kohonen, 1995) in which the training set is used to define the initial prototypes and the minimal distance to one of the prototypes is used to assign the classes. From SBM perspective it is clear that LVQ may be combined with various weighting schemes and the probability of classification may be calculated using more than a single neighbor.

The Feature Space Mapping (FSM) neurofuzzy model is based on separable (rather than radial) weighting functions (Duch and Diercksen, 1995). FSM may use many localized transfer functions, including Gaussian, conical, trapezoidal or rectangular functions. These transfer functions may again be understood as the weighting functions for prototypes localized in the neighborhood of a query vector $\mathbf{X}$. Thus, FSM may be regarded either as a specific realization of the SBM scheme or as an adaptive fuzzy logic rule-based system. A whole class of the fuzzy if-then rule systems is equivalent to the soft-weighted k-NN (Kuncheva and Bezdek, 1997, 1999).

An important problem with localized description of the data by RBF and similar methods concerns the representation of oblique probability distributions of the classes. A solution creating oblique probability distributions in N-dimensional space using only N parameters has been described quite recently (Duch and Jankowski, 1999). Oblique decision borders in SBM are obtained by rotation of the local coordinate system in which distances are computed. It is sufficient to use a rotation matrix with scaling factors $R_{ii} = s_i$ on the diagonal and rotation parameters $R_{ii+1} = \beta_i$ as the only off-diagonal element.

Relation of the SBM framework to other neural models has been discussed in details in Duch et al. (2000a). Multi-layered perceptrons, although related more to discrimination rather than clusterization methods, may also be regarded from SBM perspective if the input vectors are normalized – this may always be done in an extended feature space, adding one additional feature. Weights in such networks play the role of reference vectors and sigmoidal transfer functions play

in the form:

$$\sigma(\mathbf{W}\cdot\mathbf{X}) = \sigma\left(\frac{1}{2}(\|\mathbf{W}\|^2 + \|\mathbf{X}\|^2 - \|\mathbf{W}-\mathbf{X}\|^2)\right)$$
$$= \sigma(d_0 - D(\mathbf{W},\mathbf{X})) \tag{36}$$

where $D(\mathbf{W},\mathbf{X})$ is proportional to the square of Euclidean distance. Using $\sigma(d_0 - D(\mathbf{W},\mathbf{X}))$ activation functions with different distance functions leads to new type of neural networks (D-MLP networks) with additional non-linear parameters in the distance functions.

3.5. Neural k-NN generalizations

Neural realization of the 1-NN rule for binary patterns is ensured by the Hamming network (Lippmann, 1987, Floreen, 1991). An alternative approach is to build a network with hidden nodes realizing the **hard sphere** transfer functions, i.e. $\Theta(r - d(\mathbf{X},\mathbf{R}))$, where Θ is the Heaviside threshold function, r is the radius of the sphere and $d(\mathbf{X},\mathbf{R})$ is the distance between the vector $\mathbf{X}$ and the reference (training) vector $\mathbf{R}$. The output units for each class C_i sum the incoming signals from all active hidden nodes belonging to that class. The number N_i of units assigned to a class C_i in the radius r from the new vector $\mathbf{X}$ allows to compute the probability of classification $p(C_i|\mathbf{X}) = N_i/\sum_l N_l$. From the geometrical point of view in the input space a hard sphere is assigned to each reference vector, labeled by the name of its class, and the output unit counts how many spheres of a given class reach the point $\mathbf{X}$. Neural realization of the k-NN method finds r for which the sum of all network outputs $\sum_l N_l = k$. Formally, this can be done by introducing recurrent connections and stabilizing dynamics when the "superoutput" node achieves fixed value but in software realizations it is much simpler to select the node with maximum activity.

A network generalization of the k-NN method provides more adaptive parameters and therefore should give better results. The network should use hidden nodes computing distances $D(\mathbf{X}-\mathbf{R}^m)$, where $\mathbf{R}^m$ are reference (training) vectors. The k nodes with the smallest distances output their class label $h_l(\mathbf{X};\mathbf{R}^l) = C_i$ and the remaining nodes output $h_m(\mathbf{X};\mathbf{R}^m) = 0$. The classes are numbered $i = 1,\ldots,K$. The output layer computes probabilities using the formula:

$$O(C_i|\mathbf{X};M) = \sum_l W_{il}\cdot h_l(\mathbf{X}) \tag{37}$$
$$p(C_i|\mathbf{X};M) = \frac{O(C_i|\mathbf{X};M)}{\sum_l O(C_l|\mathbf{X};M)}.$$

The weights W_{il} between the output node computing probabilities for class C_i are initialized to $W_{il} = \mathbf{S}(C_i,C_l)/C_l$, where the matrix $\mathbf{S}(\cdot)$ estimates similarity

delta. Thus, each vector that belongs to the k nearest ones or that falls into the r radius of $\mathbf{X}$ and is of the class C_l contributes to the probability of the C_i class a value $S(C_i, C_l)$. The structure of the network is shown in Fig. 2. For the cost function that should be optimized one may take:

$$E(\mathcal{T}; \mathbf{W}, k) = \sum_{\mathbf{X}} \sum_{i} \mathcal{R}(C_i, C(\mathbf{X}))(p(C_i|\mathbf{X}; M) - \delta(C_i, C(\mathbf{X})))^2 \qquad (38)$$

where the model M includes k and output weights as parameters and $\mathcal{S}(C_i, C_j)$ is the output-class similarity function (matrix). If we want to minimize the number of classification errors output probabilities should be changed into binary values by the winner-takes-all procedure.

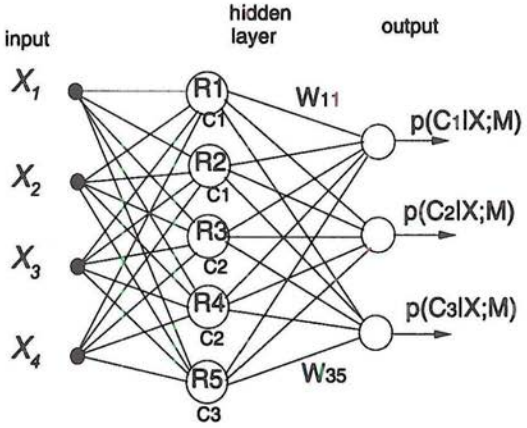

Figure 2. Network generalization of the k-NN method. The hidden nodes compute distances to reference vectors and return k values of class labels associated with the nodes, while the output nodes compute probabilities.

The output weights W_{il} are treated as adaptive parameters. Introduction of soft weighting $G(D(\cdot))$ allows to use gradient optimization methods. For many datasets this simple network should outperform many classification models. The results should be at least as good as the results of k-NN, which came out to be the best algorithm for image classification and a few other applications in the Statlog study (Michie et al., 1994).

A single neuron provides discriminating hyperplane that may be replaced by one reference vector. Position of this reference vector should be adapted to the data. Using different Minkowski distance functions dramatically changes the shape of decision borders. Using one prototype $\mathbf{R}_i$ per class (i.e. one hidden node) the class membership is decided by the discriminant function:

where θ is a threshold. The three adaptive parameters, W_1, W_2, θ, and the positions of two prototype vectors provide quite flexible decision borders in the two class problem (Fig. 3 shows an example). If more reference vectors are required the output node computing the discriminant function sums over prototypes for each class:

$$z(\mathbf{X}) = \sum_{l \in C_1} W_l D(\mathbf{X}, \mathbf{R}_l) - \sum_{l \in C_2} W_l D(\mathbf{X}, \mathbf{R}_l) - \theta. \tag{40}$$

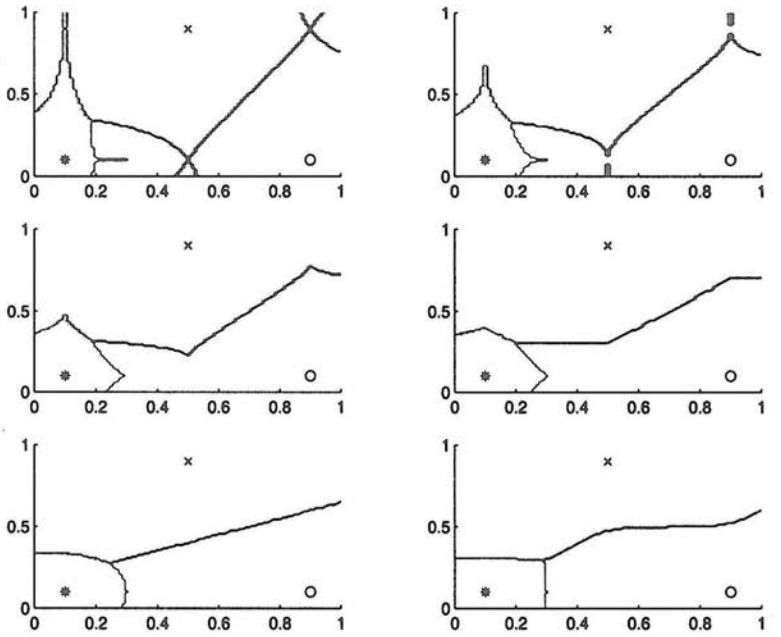

Figure 3. Decision borders for various exponents of Minkowski distance function in the nearest neighbor method for $\alpha = 0.1, 0.3, 0.7, 1, 2, 8$. Weight of the first prototype is three times larger than other weights.

Scaling of the whole sum, instead of scaling the influences of individual reference vectors, is a simple way to reduce the number of adaptive parameters used by the system. One option worth investigation is to use a simple gradient optimization for weights and thresholds, and the search based techniques for non-linear scaling parameters.

4. Discussion and related work

An overview of the similarity-based framework, with discussion of various pro-

very rich and has deep connections with many well known classification models developed by pattern recognition and neural network communities. Although this paper focused on classification methods, heteroassociation, pattern completion, and approximation problems may also be treated by similar methods. Numerous improvements of various aspects of the SBM framework have been discussed, including: a new method to convert symbolic features into numerical features, a method to find the missing data, novel functions for evaluation of similarity, methods of reference vector selection, method of improving the base rate, novel weighting functions, feature selection, combination of several models using *a priori* knowledge, connection with neural networks and neural-like realizations of SBM.

The major contribution of this paper is the change of focus from a single model to the search in space of all possible models belonging to a common framework. Starting from the simplest models new procedures and parameters are added at each stage, creating more complex models and selecting those that give the highest improvement of accuracy. In effect a best-first search (or a beam search) is performed in the space of all possible models. The final model selected may involve a combination of parameterizations and procedures corresponding to known classification model or to a new method. Although we have no space here to present experimental results it may be worthwhile to mention that preliminary implementation of the ideas presented here allowed us (Grudziński and Duch, in preparation) to obtain the best results in classification of more than half of the 20 datasets used in the Statlog project (Michie et al., 1994) and we are quite confident that results that are statistically indistinguishable from the best ones may be obtained for the remaining datasets using SBM methods.

The work presented here is related to many developments in computational intelligence, trying to integrate numerous efforts in different branches of this field. A survey of the nearest neighbor methods has been published (Dasarathy, 1990) but many aspects of SBM are not discussed there. Wettschereck and Dietterich (1997) have tested several methods of variable k selection in different input regions (multi-model approach in our terminology), using the k-NN method. Surprisingly, the results for real datasets were sometimes worse than for k-NN with a single k, except in cases where two datasets were mixed together, each requiring quite different k for good classification. Perhaps they have approached the problem in a wrong way since a proper combination of local models should always give a better result (or at least the same result) as a single best model does.

Lowe (1995) introduced the Variable Kernel Classifier based on Variable-kernel Similarity Metric (VSM). In fact, his approach is a version of the RBF method. It is based on optimization of distance scaling factors for each feature, equivalent to optimization of Gaussian dispersions. His formula for probability is:

$$p(C_i|\mathbf{X};M) = \frac{\sum_{m=1}^{k} G(\mathbf{X},\mathbf{R}^m)p_i(\mathbf{R}^m)}{\sum_{m=1}^{k} G(\mathbf{X},\mathbf{R}^m)} \tag{41}$$

where $p_i(\mathbf{R}^m)$ is the probability that the reference vector $\mathbf{R}^m$ belongs to class C_i and $G(\cdot)$ is the weighting function; this may be taken as 0 or 1, according to known classes, or estimated for a given k using the leave-one-out procedure. Optimization is done by assigning k neighbors to each reference vector. After gradient-based optimization k-neighbors are selected again and optimization repeated, if necessary. Lowe reports that deleting reference vectors from regions where classification is unambiguous (if all neighbors assign the reference vector proper class with $p > 0.6$) actually improved generalization slightly. Tests on the noisy XOR problem with 4 inputs showed the ability of VSM method to select relevant inputs and assign them larger weights.

Yang et al. (1998) introduced a constructive neural method called *DistAI* based on inter-pattern distances. A single hidden layer of hard-sphere weighting functions is constructed, each covering as many vectors of a single class as possible. This algorithm is similar to the network realization of the Restricted Coulomb Energy algorithm (Wasserman, 1993) but the neural units realize the difference between two concentric spheres of different radiuses rather than a single sphere (formally two radiuses, called "thresholds", are defined for each neural unit). The algorithm also checks for a single attribute that separates the largest number of vectors from a single class. Distance matrix between all vectors is computed once and sorted in the ascending order; each row i corresponds to distances from the vector $\mathbf{X}^{(i)}$. Spherical functions realized by neurons of the *DistAI* network are equivalent to the reference vectors selected from the training set by checking each vector (row) and counting the number of training vectors belonging to a single class (it may be different than the class of the selected vector), i.e. checking in a row how many consecutive entries are from a single class. The vector for which maximum has been found defines the center of the new function and the minimal and the maximal radiuses are defined using the closest and the furthest vector from this center. All patterns correctly covered by the new function are removed from the training set and the next hidden neuron is defined. Since new functions may overlap with the old ones each new neuron has weights that are by a factor of two smaller than the previous one. The worst case complexity of this algorithm is of the order $O(N)$, where N is the number of patterns. Although Yang et al. (1998) report very good results for many datasets the decision borders of such classifier are far from natural.

From these papers and from the preliminary numerical experiments with SBM methods a few conclusions may be drawn. Scaling of individual features is very important and can bring substantial gains in accuracy as well as reduce the number of features. Selection of a fixed number of neighbors works usually better than optimization of one radius in which the number of neighbors is counted. If the optimal number of neighbors is small, weighting procedures do not contribute significantly to accuracy. Much better results are probably achieved if local weighting functions are introduced, similarly as in the RBF, where adaptation of individual dispersions is of great importance, or if the α-

Hastie and Tibshirani (1996) write about adaptive k-NN classification from the linear discriminant point of view, advocating the use of several local metrics in different areas of the input space, instead of just one. Friedman (1994) proposed an interesting way of adapting the metric based on a tree-structure interactive partitioning of the data. Laaksonen and Oja (1996) proposed to improve the k-NN reference vectors using LVQ techniques. Atkenson, Moor and Schaal (1997) discuss locally weighted regression techniques, minimal distance methods with various metric and kernel functions applied to approximation problems.

All these and many more proposals may be accommodated in the general framework presented here. Identification of the best combination of procedures and adaptive parameters should allow for improvement of results achieved by the nearest neighbor as well as neural classifiers. Many possibilities of creating fuzzy k-NN models remain to be explored (see Bezdek et al., 1986). Performance of various methods described here (as well as any other classification methods) depends on the nature of the data given for classification and remains a subject of further empirical study. We have already developed and tested many variants of SBM methods described here and we shall deal with the empirical evaluation of these models separately (Grudziński and Duch, in preparation). Our main goal is to develop software that will automatically construct a series of models of growing complexity and growing accuracy, combining various procedures described in this paper.

Acknowledgments: Support by the Polish Committee for Scientific Research, grant no. 8 T11C 006 19, is gratefully acknowledged.

References

AHA, D.W. (1998) Feature weighting for lazy learning algorithms. In: *Feature Extraction, Construction and Selection: A Data Mining Perspective*, Liu, H. and Motoda, H., eds., Kluwer, Norwell, MA.

AHA, D., KIBLER, D. and ALBERT, M. (1991) Instance-based learning algorithms. *Machine Learning*, **6**, 37–66.

ATKENSON, C.G., MOOR, A.W. and SCHAAL, S. (1997) Locally weighted learning. *Artificial Intelligence Review*, **11**, 75–113.

BEZDEK J.C., CHUAH, S.K. and LEEP, D. (1986) Generalized k-nearest neighbor rule. *Fuzzy Sets and Systems*, **18**, 237–256.

BISHOP, C.M. (1995) *Neural Networks for Pattern Recognition*. Oxford University Press.

BREIMAN, L. (1998) Bias-Variance, regularization, instability and stabilization. In: *Neural Networks and Machine Learning*, Bishop, C., ed., Springer.

CHIU, D.K.Y. and KAVANAUGH, F.E. (1997) The ck-nearest neighbor distance network: a network using class boundary feature distances. *ICONIP'97*,

DASARATHY, B.V., ed. (1990) *Nearest neighbor norms: NN Pattern Classification Techniques.* IEEE Computer Society Press, Los Alamitos, California.

DUCH, W. (1998) A framework for similarity-based classification methods. *Intelligent Information Systems VII, Malbork, Poland*, pp. 288–291.

DUCH, W., ADAMCZAK, R. and DIERCKSEN, G.H.F. (2000) Neural Networks and Similarity-Based Methods. *Applied Mathematics and Computer Science*, **10**, 101–120.

DUCH, W., ADAMCZAK, R. and GRĄBCZEWSKI, K. (1997) Extraction of crisp logical rules using constrained backpropagation networks. *Proc. of International Joint Conference on Neural Networks (IJCNN'97), Houston, Texas*, pp. 2384–2389.

DUCH, W., ADAMCZAK, R., GRĄBCZEWSKI, K., ŻAL, G. and HAYASHI, Y. (1999) Fuzzy and crisp logical rule extraction methods in application to medical data. In: *Fuzzy systems in Medicine*, Szczepaniak, P.S., Lisboa, P.J.G. and Kacprzyk, J., eds., Physica-Verlag, Springer, 2000, pp. 593–616.

DUCH, W. and DIERCKSEN, G.H.F. (1995) Feature Space Mapping as a universal adaptive system. *Computer Physics Communication*, **87**, 341–371.

DUCH, W. and GRUDZIŃSKI, K. (1998) A framework for similarity-based methods. *2nd Polish Conference on Theory and Applications of Artificial Intelligence, Łódź*, pp. 33–60.

DUCH, W. and GRUDZIŃSKI, K. (1998a) A framework for similarity-based methods. *2nd Polish Conference on Theory and Applications of Artificial Intelligence, Łódź*, pp. 33–60;

DUCH, W. and GRUDZIŃSKI, K. (1999) The weighted k-NN method with selection of features and its neural realization. *4th Conference on Neural Networks and Their Applications, Zakopane*, pp. 191–196.

DUCH, W. and GRUDZIŃSKI, K. (1999a) Search and global minimization in similarity-based methods. *International Joint Conference on Neural Networks (IJCNN'1999), Washington*, paper no. 742.

DUCH, W. and GRUDZIŃSKI, K. (1999b) Weighting and selection of features in Similarity-Based Methods. *Intelligent Information Systems VIII, Ustroń, Poland*, pp. 32–36.

DUCH. W., GRUDZIŃSKI, K. and DIERCKSEN, G.H.F. (1998) Neural minimal distance methods. *World Congress of Computational Intelligence, May 1998, Anchorage, Alaska, IJCNN'98 Proceedings*, pp. 1299–1304.

DUCH, W. and JANKOWSKI, N. (1999) New neural transfer functions. *Neural Computing Surveys*, **2**, 639–658.

FLOREEN, P. (1991) The convergence of Hamming memory networks. *IEEE Transactions Neural Networks* **2**, 449–457.

FRIEDMAN, J.H. (1994) *Flexible metric nearest neighbor classification.* Techni-

FUCHS, M. (1996) *Optimized nearest-neighbor classifiers using generated instances.* LSA-96-02E Technical Report, Learning Systems & Applications Group, University of Kaiserslautern, Germany.

GRĄBCZEWSKI, K. and DUCH, W. (1999) A general purpose separability criterion for classification systems. *4th Conf. on Neural Networks and Their Applications, Zakopane*, pp. 203–208.

GRUDZIŃSKI, K. and DUCH, W. (2000) SBL-PM: A Simple Algorithm for Selection of Reference Instances for Similarity-Based Methods, *Intelligent Information Systems IIS'2000*, Physica-Verlag (Springer), pp. 99–108.

HASTIE, T. and TIBSHIRANI, R. (1996) Discriminant adaptive nearest neighbor classification. *IEEE PAMI*, **18**, 607–616.

HAYKIN, S. (1994) *Neural Networks. A Comprehensive Foundation.* MacMillian, New York.

KOHONEN, T. (1995) *Self-organizing maps.* Springer-Verlag, Berlin.

KRISHNAIAH, P.R. and KANAL, L.N., eds. (1982) *Handbook of statistics 2: classification, pattern recognition and reduction of dimensionality.* North-Holland, Amsterdam.

KUNCHEVA, L.I. and BEZDEK, J.C. (1997) A fuzzy generalized nearest prototype classifier. In: *Proc. 7th IFSA World Congress, Prague, Czech Republic*, vol. III, pp. 217–222.

KUNCHEVA, L.I. and BEZDEK, J.C. (1999) Presupervised and Postsupervised Prototype Classifier Design. *IEEE Transactions on Neural Networks*, **10**, 1142–1152.

LAAKSONEN, J. and OJA, E. (1996) Classification with Learning k-Nearest Neighbors. In: *Proc. of ICNN'96*, WASHINGTON, *D.C.*, pp. 1480–1483.

LIPPMANN, R.P. (1987) An introduction to computing with neural nets. *IEEE Magazine on Acoustics, Signal and Speech Processing*, **4**, 4–22.

LOWE, D.G. (1995) Similarity metric learning for variable-kernel classifier. *Neural Computation*, **7**, 72–85.

MERTZ, C.J. and MURPHY, P.M. (1999) UCI repository, http://www.ics.uci.edu/pub/machine-learning-databases.

MICHALSKI, R. (1999) AQ-PM: A System for Partial Memory Learning. *Intelligent Information Systems VII, Ustroń, Poland, 14–18 June 1999*, pp. 70–79.

MICHIE, D., SPIEGELHALTER, D.J. and TAYLOR, C.C. (1994) *Machine learning, neural and statistical classification.* Elis Horwood, London.

MITCHELL, T.M. (1997) *Machine Learning.* McGraw-Hill.

REILLY, D.L., COOPER, L.N. and ELBAUM, C. (1982) A neural model for category learning. *Biological Cybernetics*, **45**, 35–41.

RIPLEY, B. (1996) *Pattern Recognition and Neural Networks.* Cambridge University Press.

ROHWER, R. and MORCINIEC, M. (1996) A Theoretical and Experimental Account of n-tuple Classifier Performance. *Neural Computation*, **8**, 657–670.

RUBIN, D.B. (1996) Multiple imputation after 18+ years. *J. of the American*

SCHÖLKOPF, B., BURGES, C. and SMOLA, A. (1998) *Advances in Kernel Methods: Support Vector Machines*, MIT Press, Cambridge, MA.

SILVERMAN, B.W. (1986) *Density estimation for statistics and data analysis.* Chapman and Hall, London.

SIMARD, P., LECUN, Y. and DENKER, J. (1993) Efficient pattern recognition using a new transformation distance. *Advances in Neural Information Processing Systems 5 (NIPS'5), San Mateo, CA*, pp. 50–58.

STANFILL, C. and WALTZ, D. (1986) Toward memory-based reasoning. *Communications of ACM*, **29**, 1213–1228.

WALTZ, D.L. (1995) Memory-based reasoning. In: *The Handbook of Brain Theory and Neural Networks*, Arbib, M.A., ed., MIT Press, pp. 568–570.

WASSERMAN, P.D. (1993) *Advanced methods in neural networks.* Van Nostrand Reinhold.

WETTSCHERECK, D. and AHA, D.W. (1995) Weighting Features. In: *1st Int. conf. on Case-based Reasoning (ICCBR-95), Lisbon, Portugal*, Springer-Verlag.

WETTSCHERECK, D., AHA, D.W. and MOHRI, T. (1997) A Review and Empirical Evaluation of Feature Weighting Methods for a Class of Lazy Learning Algorithms. *Artificial Intelligence Review*, **11**, 273–314.

WETTSCHERECK, D. and DIETTERICH. T.G. (1997a) Locally adaptive nearest neighbor algorithms. *Advances in Neural Information Processing Systems 6 (NIPS'6), San Mateo, CA*, pp. 184–191.

WILSON, D.R. and MARTINEZ, T.R. (1997) Improved heterogeneous distance functions. *Journal of Artificial Intelligence Research*, **6**, 1–34.

YANG, J. PAREKH, R. and HONAVAR, V. (1998) DistAI: an inter-pattern distance-based constructive learning algorithm. *World Congress on Computational Intelligence, Anchorage, Alaska*, pp. 2208–2213.