A framework for similarity-based methods.

Włodzisław Duch, Karol Grudziński
Department of Computer Methods, Nicholas Copernicus University,
Grudziądzka 5, 87-100 Toruń, Poland; E-mail:duch,kagru@phys.uni.torun.pl

Abstract

Similarity-based methods (SBM) are a generalization of the minimal distance (MD) methods which form a basis of many machine learning and pattern recognition method. Investigation of similarity leads to a fruitful framework in which many classification methods are accommodated. Probability $p(C|X;M)$ of assigning class C to vector X, given the classification model M depends on adaptive parameters of the model and procedures used in calculation, such as: the number of reference vectors taken into account in the neighborhood of X, the maximum size of the neighborhood, parameterization of the similarity measures, the weighting function estimating contributions of neighboring reference vectors, the procedure used to create a set of reference vectors from the training data, the total cost function minimized at the training stage and the kernel function, scaling the influence of the error on the total cost function. SBM may include several models M_l and an interpolation procedure to combine results of a committee of models. Neural networks, such as the Radial Basis Function (RBF) and the Multilayer Perceptrons (MLPs) models, are included in this framework as special cases. Many new versions of similarity based methods are derived from this framework. A few of them have been tested on a real-world data sets and very good results have been obtained.

Keywords

k-NN, minimal distance methods, similarity, neural networks, classification.

I. Introduction

Recently one of us [1] has presented a general framework for minimal distance methods. This framework is extended here, leading to a new versions of neural methods and similarity distance methods. Classification is one of the most important applications of neural systems. A review of different approaches to classification and comparison of performance of 20 methods on 20 real world datasets has been done within the StaLog European Community project [2]. 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 [3]. 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. 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 [3], are among the best. Such minimal distance methods have natural neural realizations. A whole family of similarity-based methods is defined in this paper, allowing to choose the best method for a given problem among members of one family. Results of several preliminary tests on a real-world datasets for a few new versions of similarity-based methods are presented.

Some of the simplest classification algorithms applicable to pattern recognition problems are based on the k-nearest neighbor (k-NN) rule [4]. This approach is so important that in artificial intelligence it is referred to as the instance based learning, memory based learning or case based learning [5]. Each training data vector is labeled by the class it belongs to and is treated as a reference vector. During classification k nearest reference vectors 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 class C_i is $p(C_i|\mathbf{X}) = N_i/k$. In the simplest case of $k = 1$ only the 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 [4]. 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 (although surprisingly very few computer programs for k-NN are around). One problem is the computational complexity of the actual classification, demanding for n reference vectors calculation of $\sim n^2$ distances and finding

k smallest distances. Although Laaksonen and Oja [6] 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. Even without any speedup of computations 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 (cf. [7] 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 [8] 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 minimal distance methods presented here have natural neural-network type realizations we will concentrate more on presentation of the general framework and testing of specific methods derived from this framework, rather than on the network implementation issues since at this initial stage implementation is of secondary importance. In the next section general framework is presented and relations between neural and minimal distance methods explained. In the third section results of tests on real datasets are given. A short discussion is presented in the last section.

II. A framework for the similarity-based methods

The problem of classification is stated as follows: given a set of class-labeled training vectors $\{\mathbf{X}^p, \mathbf{C}(\mathbf{X}^p)\}$, $p = 1..K$, where $\mathbf{C}(\mathbf{X}^p)$ is the class of $\mathbf{X}^p$, and a vector $\mathbf{X}$ of an unknown class, use the information provided in the distance $d(\mathbf{X}, \mathbf{X}^p)$ to estimate probability of classification $p(C_i|\mathbf{X}; M)$, where M describes the classification model used (parameter values and procedures employed). The empirical risk matrix $R(C_i, C_j)$ measures the risk of misclassification (in the simplest case $R(C_i, C_j) = \delta_{ij}$) and the error function that should be minimized is:

$$E(M) = \sum_p \sum_k R(C(\mathbf{X}_p), C_k) p(C_k|\mathbf{X}_p; M) \tag{1}$$

where p runs over all vectors in the training dataset, k runs over all classes and $C(\mathbf{X}_p)$ is the true class of the vector $\mathbf{X}_p$. Parameters and procedures defining the model M should be found that minimize the probability of misclassification and thus maximize the probability of correct classification (although there are some differences between the two approaches they will be ignored here).

A general model of an adaptive system used for classification may include all or some of the following:
$M = \{k, d(\cdot; r), G(d(\cdot)), \{\mathbf{R}^n\}, \{p_i(R)\}, E[\cdot], K(\cdot)\}$, where
k is the number of reference vectors taken into account in the neighborhood of $\mathbf{X}$;
$d(\cdot; r)$ is the function used to compute similarities (usually distances);
r is the maximum size of the neighborhood considered;
$G(d(\mathbf{X}, \mathbf{X}^p))$ is the weighting function estimating contribution of reference vector $\mathbf{X}^p$ to the classification probability;
$\{\mathbf{R}^n\}$ is the set of reference vectors created from the set of training vectors $\{\mathbf{X}^p\}$ by some procedure;
$p_i(R)$ is the set of class probabilities for each reference vector
$E[\cdot]$ is the total cost function that is minimized at the training stage;
$K(\cdot)$ is a kernel function, scaling the influence of the error, for a given training example, on the total cost function.

In addition an adaptive system may include several models M_l and an interpolation procedure to select between different models or combine results of a committee of models. Various selections of parameters and procedures in the context of network computations lead to different versions of similarity-based classification methods. If the similarity function has metric properties the similarity-based method becomes the minimal distance-based (MD) method, but here we prefer to include also functions that do not have metric properties. Several variants of similarity-based (SB) classification methods are presented below.

k-NN networks.

In the simplest version $p(C_i|\mathbf{X}; M)$ is parameterized by $p(C_i|\mathbf{X}; k, d(\cdot), \{\mathbf{X}^n\}\})$, and the k-NN method, in which the whole training dataset is used as the reference set, is obtained. The distance function $d(\mathbf{X}, \mathbf{X}^n)$ is a hard-sphere metric function with radius such that exactly k neighboring vectors $\mathbf{X}^n$ fall inside it. The type of the metric (see below) and k are the only parameters that should be optimized. 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 L-class problem $k = 1, L+1, 2L+1, ...$ 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 braking the tie and e) giving probabilities instead of yes-no decisions. Details of the k-NN procedure are rarely given in practice and it is not clear which of the five possibilities is used. In our experience adding more vectors to break the tie is preferable although differences in classification accuracy are sometimes negligible.

The simplest error function used in optimization of k and the selection of the type of metric d is:

$$\begin{aligned} E(\mathbf{X}; k, d) &= \sum_{p=1}^{K} (1 - \delta(C(\mathbf{X}^p), C_j(\mathbf{X}^p)) \\ C_j(\mathbf{X}^p) &\leftarrow \max_j \ p(C_j|\mathbf{X}^p; M) \end{aligned} \tag{2}$$

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). In some problems classes have natural similarity and can be ordered in such a way that a different error is associated with predictions of similar classes than with the predictions of quite different classes. This is done either by summing the squares of $(C(\mathbf{X}^p) - C_j(\mathbf{X}^p))^2$ or in a more universal way by adding a loss (risk) matrix (with zeros on the diagonal) to the error function:

$$E_R(\mathbf{X}; k, d) = \sum_{p=1}^{K} R(C(\mathbf{X}^p), C_j(\mathbf{X}^p)) \tag{3}$$

Neural realization of the 1-NN rule for binary patterns is afforded by the Hamming network [8]. An alternative approach 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 sum the incoming signals from all active hidden nodes belonging to that class. The number N_i of such units in the radius r from the new vector $\mathbf{X}$ allows to compute the probability of classification $p(C_i|\mathbf{X}) = N_i / \sum_j N_j$. 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 k-NN method finds r for which the sum of all network outputs $\sum_j N_j = k$. Formally this can be done by introducing recurrent connections and stabilizing dynamics when the “superoutput” node achieves fixed value.

***r*-NN algorithm**.

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 $p(C_i|\mathbf{X}; r) = N_i / \sum_j N_j$, is then optimized using the leave-one-out method or a validation set. The hard sphere transfer functions should be used in the 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 radiuses r_i for each reference vector instead of one universal radius in the input space improves the method 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 [9] which may be treated as the hard limit approximation of the Gaussian-based RBF network. If no neighbors are found around the training vector $\mathbf{X}$ 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 [6]), but voting methods for the committees of classifiers were used with success [10].

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.

Soft weighting *k*-NN and NN-*r* algorithms.

Nearest reference vectors should influence probabilities of classification more strongly than farther laying neighbors. Changing the hard sphere transfer functions into smoother functions allows to include such weights. The favorite fuzzy logic membership function is **the conical radial function**: 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:

$$\begin{aligned} p(C_i|\mathbf{X};r) &= \frac{\sum_{n \in C_i} G(\mathbf{X};\mathbf{R}^n, r)}{\sum_n G(\mathbf{X};\mathbf{R}^n, r)}; \\ G(\mathbf{X};\mathbf{R}, r) &= \max\left(0, 1 - \frac{d(\mathbf{X},\mathbf{R})}{r}\right) \end{aligned} \tag{4}$$

Here $G(\mathbf{X};\mathbf{R}, r)$ is the weight estimating contribution of reference vector at the distance $d(\mathbf{X}, \mathbf{R})$. In the soft NN-$r$ algorithm the r parameter is optimized. Radial Basis Function (RBF) networks using Gaussian or inverse multiquadratic transfer functions are a particular example of the soft weighting minimal distance algorithm. Other useful weighting functions include a combination of two sigmoidal functions: $\sigma(||\mathbf{X} - \mathbf{R}^n|| - r) - \sigma(||\mathbf{X} - \mathbf{R}^n|| + r)$.

One may also try to use variable r equal to the distance to the k-th neighbor and use the weighting function for the vectors inside this radius. If r_k is the distance to the k-th neighbor and $r_k \geq r_i, i = 1..k-1$ then a conical weighting function $G(d) = 1 - d/\alpha r_k, \alpha > 1$ 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. Minimal distance system with optimized α cannot be less accurate than k-NN. The effect of weighting is more pronounced for larger k values. If k is taken as the number of all remaining training vectors and α is optimized the results should be close to the r-NN method but if both k and α are optimized the results should be better.

The cost function is either a classification error (as for the hard-distance case) or – since continuos output values are provided – minimization of risk for overall classification:

$$E_R(M) = \sum_{i,\mathbf{X}} R(C_i, C(\mathbf{X})) \left[p(C_i|\mathbf{X};M) - \delta(C_i, C(\mathbf{X}))\right]^2 \tag{5}$$

where the sum runs over all training vectors $\mathbf{X}$, $C(\mathbf{X})$ is the true class of vector $\mathbf{X}$, $R(C_i, C_j)$ is the risk matrix, and M specifies parameters of the classifier. To minimize the leave-one-out error the sum runs over all training examples $\mathbf{X}^p$ and the model used to specify the classifier should not contain the $\mathbf{X}^p$ vector in the reference set while $p(C_i|\mathbf{X}^p)$ is computed.

Wettschereck and Aha [24] propose the hyperbolic weighting scheme:

$$p(C|X;M) = \frac{\sum_{R \in O_k(X)} \delta(C(X), C) 1/(d(X,R) + \varepsilon)}{\sum_{R \in O_k(X)} 1/(d(X,R) + \varepsilon)} \tag{6}$$

where $O_k(X)$ is the neighborhood of X containing k reference vectors R.

Parameterization of similarity measures

Calculation of similarities is most often reduced to the Euclidean metric for continuos inputs and Hamming metric for binary inputs. In a more general approach let us first define one-dimensional similarity functions $\delta_i(A_i, B_i)$. This may be:

$$\delta_i(A_i, B_i) = A_i - B_i \quad \text{a simple difference} \tag{7}$$

$$\delta_i(A_i, B_i) = |A_i - B_i| \quad \text{an absolute value of the difference} \tag{8}$$

$$\delta_i(A_i, B_i) = \frac{A_i - B_i}{Max_i - Min_i} \quad \text{renormalized difference} \tag{9}$$

$$\delta_i(A_i, B_i) = \frac{A_i - B_i}{4\sigma_i} \quad \text{standardized difference} \tag{10}$$

$$\delta_i(A_i, B_i) = \delta(A_i, B_i) \quad \text{overlap difference} \tag{11}$$

where in the last case Kroneker delta is used and $A_i = X_i^{(p)}$, or it is a preprocessed value of the i-th component of the $\mathbf{X}^{(p)}$ vector. One-dimensional similarity functions may also be taken as the probabilistic value difference:

$$\delta_i^{VD}(A_i, B_i)^\alpha = \sum_j \left|p(C_j|A_i) - p(C_j|B_i)\right|^\alpha = \sum_j \left|\frac{N(C_j|A_i)}{N(A_i)} - \frac{N(C_j|B_i)}{N(B_i)}\right|^\alpha \tag{12}$$

where $N(A_i)$ is the number of vectors in the dataset with the attribute i equal to the value A_i and $N(C_j|A_i)$ is the number of such vectors belonging to the class C_j. Minkowski's metric involves one exponent α which has already been included in the definition of the value difference. Typical distance function compute:

$$d(\mathbf{A}, \mathbf{B})^\alpha = \sum_i^N \delta_i(A_i, B_i)^\alpha \tag{13}$$

$$d(\mathbf{A}, \mathbf{B}) = \max_i \delta_i(A_i, B_i) \quad \text{Maximum Value} \tag{14}$$

In memory-based reasoning the Modified Value Difference Metric (MVDM) has gained popularity [7]. The distance between two N-dimensional vectors $\mathbf{A}, \mathbf{B}$ with discrete (for example symbolic) elements, in a K class problem, is computed using conditional probabilities:

$$d^{MVDM}(\mathbf{A}, \mathbf{B}) = \sum_i^N \sum_j^K (p(C_j|A_i) - p(C_j|B_i)) = \sum_i^N \delta_i^{VD}(A_i, B_i) \tag{15}$$

Similarity is defined here 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 all attributes. Generalization for continuos values requires a set of probability density functions $p_{ij}(x)$, with $i = 1..K$, $j = 1..N$. This distance function may be used for symbolic values and combined with other distance functions for continuos attributes.

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

$$d(\mathbf{A}, \mathbf{B}; s)^\alpha = \sum_i^N s_i \delta_i (A_i, B_i)^\alpha; \quad s_i \geq 0 \tag{16}$$

Euclidean metric corresponds to $\alpha = 2$, which is completely isotropic, and Manhattan metric to $\alpha = 1$, which is less sensitive to the directions parallel to the axis than to the directions between the axis. In fact the unit contour is a circle for Euclidean and a square (with vertices in $(0, \pm 1)$ and $(\pm 1, 0)$), approaching a square with vertices at $(\pm 1, \pm 1)$ for large α and a concave 4-arm star for α going to zero. Many methods of selecting optimal scaling factors for features were reviewed by Wettschereck et al. [24].

Defining the scalar product and the norm:

$$\langle \mathbf{A}|\mathbf{B} \rangle = \sum_{i=1}^N A_i B_i; \quad ||\mathbf{A}||^2 = \langle \mathbf{A}|\mathbf{A} \rangle \tag{17}$$

several other distance functions are defined:

$$d_c(\mathbf{A}, \mathbf{B}) = 1 - \frac{\langle \mathbf{A}|\mathbf{B} \rangle}{||\mathbf{A}||\,||\mathbf{B}||} \quad \text{Cosine distance} \tag{18}$$

$$d_d(\mathbf{A}, \mathbf{B}) = 1 - \frac{2\langle \mathbf{A}|\mathbf{B} \rangle}{||\mathbf{A}||^2 + ||\mathbf{B}||^2} \quad \text{Dice distance} \tag{19}$$

$$d_J(\mathbf{A}, \mathbf{B}) = 1 - \frac{\langle \mathbf{A}|\mathbf{B} \rangle}{||\mathbf{A}||^2 + ||\mathbf{B}||^2 - \langle \mathbf{A}|\mathbf{B} \rangle} \quad \text{Jaccard distance} \tag{20}$$

$$d_C(\mathbf{A}, \mathbf{B}) = \frac{\delta_i(A_i, B_i)}{\delta_i(A_i, -B_i)} \quad \text{generalized Camberra distance} \tag{21}$$

Additional parameters that may be optimized are either global or local (different for each reference vector). In the simplest version of the RBF algorithm with Gaussian functions only one parameter – dispersion – is optimized. Independent optimization of all N components of dispersion vector has the same effect as optimization of the scaling factors g_i in soft-weighted NN-r method. Scaling is the simplest way of pre-processing the attributes. The scaling factors facilitate feature selection in an automatic way. Admitting only $s_i = 0, 1$ allows for simplified optimization of the scaling factors for feature selection.

In some applications similarities are not symmetric. The simplest extension to non-symmetric similarity function is by introducing different scaling factors, depending on the sign of $A_i - B_i$ difference, for example:

$$d_n(\mathbf{A}, \mathbf{B}; g)^\alpha = \sum_i^N (\max(0, s_{i+}(A_i - B_i)) - \min(0, s_{i-}(A_i - B_i)))^\alpha; \quad s_i \geq 0 \tag{22}$$

This function provides $2N$ adaptive parameters. Mahalanobis distance is obtained by applying a particular linear transformation to the input vectors. Alternatively, a metric tensor $G_{ij} = G_{ji}$ is introduced, providing $N(N+1)/2$ adaptive parameters:

$$d(\mathbf{A}, \mathbf{B}; \mathbf{G})^2 = \sum_{i,j}^N G_{ij}(A_i - B_i)(A_j - B_j) \tag{23}$$

Any adaptive system may provide a distance function for SB methods. For example [11], a typical MLP network may be trained on the differences of pairs of vectors $\{A_i - B_i\}$, giving at the output the distance between the classes $||C(\mathbf{A}) - C(\mathbf{B})||$. The output of the neural network is then used in k-NN or other SB procedure. A better way is to input to MLP both $\mathbf{A}$ and $\mathbf{A} - \mathbf{B} = \{d^i(A_i) - d^i(B_i)\}$ vectors, where $d^i(\cdot)$ is a set of attribute pre-processing functions (for example, scaling factors). A similarity function (it does not have all properties required from the distance function, for example it is not symmetric) $d(\mathbf{A}-\mathbf{B};\mathbf{A})$, smoothly changing between different regions of the input space, is obtained iteratively: for each training vector k nearest neighbors are selected using initial similarity estimation, and after the first epoch the process is repeated using the new similarity function.

In general one should create a distance function that minimizes in-class distances and maximizes between-class distances. A very convenient function of this sort is based on a combination of sigmoidal functions in each dimension:

$$A_i = \varphi_i(x_i;\mathbf{a}_i,\mathbf{b}_i,\mathbf{c}_i) = \sum_{j=1}^{K_i} a_{ij}\sigma(b_{ij}X_i + c_{ij}) \qquad (24)$$

where K_i determines the number of steps in the smoothed sigmoidal step function. Replacing A_i and B_i in the Minkowski's metric with these functions allows to model classes obtained with logical rules.

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.

Active selection of reference vectors.

In MD method one should avoid large number of reference vectors. Reducing the number of vectors in the reference set leads to models of lower complexity and helps to improve generalization capabilities of the classification system. K-means, dendrograms or other clusterization techniques are used to select a relatively small number of initial reference vectors close to the cluster centers. Classification accuracy is checked on the remaining set (using k-NN or r-NN algorithm) and each wrongly classified vector is moved from the training to the reference set. Variants of this approach my use a validation set to determine best candidates for the reference set.

An alternative approach that does not require initial clusterization starts from the whole training set and removes those vectors that have all k nearest vectors from the same class. These vectors are far from cluster borders and all new vectors in their neighborhood will be anyway unambiguously classified. This approach leads to a “hollow” cluster representation. Here one may start with a large number of neighbors k' to remove vectors near the centers of clusters first and reduce it to k in a few steps. An interesting algorithm to select good reference vectors is: run over all vectors $\mathbf{X}$, determine k nearest vectors from each class different than $C(\mathbf{X})$ and move these vectors to the reference set. This algorithm leaves only vectors near the class borders. Reducing the number of vectors in the reference set leads to models of lower complexity and helps to improve generalization capabilities of the classification system.

Parameterization of reference vectors

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 $\mathbf{R}^n$ in the neighborhood of training vector $\mathbf{X}$ should be updated as follows:

$$\mathbf{R} \leftarrow \mathbf{R} + \eta\delta_{\pm}(C(\mathbf{X}),C(\mathbf{R}))(\mathbf{X}-\mathbf{R}) \qquad (25)$$

Here η is the learning rate, slowly decreasing during training, and $\delta_{\pm}$ is $+1$ if the class $C(\mathbf{X}) = C(\mathbf{R})$ or -1 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 etc. [6]. 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 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.

Selection of the kernel function and the error function The choice of kernel function in the measure of classification error should also be discussed. In local regression based on the minimal distance approaches [12] the error function is simply

$$E(\mathbf{X};M) = \sum_i K(d(\mathbf{X}^i,\mathbf{R}^{ref}))(F(\mathbf{X}^i;M)-y^i)^2 \qquad (26)$$

where y^i are the desired values for $\mathbf{X}^i$ and $F(\mathbf{X}^i;M)$ are the values predicted by the model M; here the kernel function $K(d)$ measures the influence of the reference vectors on the total error. For example 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 and will make large errors for other values. This is not quite the same as the weighting function

$G(d)$ which is used to estimate the distance. In classification problems kernel function will determine the size of the neighborhood around the known cases in which accurate classification is required.

The cost function is either a classification error (as for the hard-distance case) or – since continuos output values are provided – minimization of risk for overall classification:

$$E_R(\mathbf{X};M) = \sum_{i,p} R(C_i, C(\mathbf{X}^p)) \left[p(C_i|\mathbf{X}^p;M) - \delta(C_i, C(\mathbf{X}^p)) \right]^2 \tag{27}$$

where the sum runs over all training vectors $\mathbf{X}$, $C(\mathbf{X})$ is the true class of vector $\mathbf{X}^p$, $R(C_i,C_j)$ is the risk matrix, and M specifies parameters of the classifier. To minimize the leave-one-out error the sum runs over all training examples $\mathbf{X}^p$ and the model used to specify the classifier should not contain the $\mathbf{X}^p$ vector in the reference set while $p(C_i|\mathbf{X}^p)$ is computed.

Additional procedures.

Feature selection:

The simplest feature selection procedure is based on excluding successively each feature, reoptimizing the model and comparing the final accuracy: if the accuracy increases the feature should be deleted. This method is quite effective and has frequently improved the results significantly while reducing the number of features to a few only. Many methods of feature selection and estimation of optimal scaling factors for features were reviewed by Wettschereck et al. [24]. These methods either iteratively optimize the scaling factors on the performance basis or assign fixed scaling factors calculating mutual information between the values of features and the class of training samples or by summing probabilities (estimated using 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.

A better (but ore costly) way to select features and at the same time to lower the complexity of the classification model is to add to the cost function an additional penalty term, such as the sum of all g_i^2 or G_{ij}^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 is an alternative but more complicate solution). Features for which the product of the scaling factors $s_i \max_{jk} |X_i^{(j)} - X_i^{(k)}|$ is small may be deleted without significant loss of accuracy – after additional optimization of the scaling factors accuracy may even increase.

Treatment of missing values:

The Value Difference Metric treats the missing value like any other symbolic value. Frequently the missing values are replaced by some averages or ignored replaced by a constant additive factor in the context of distance calculation. One method worth trying is to count the number of vectors in each class in the neighborhood of the partially known vector in the subspace of known features. The largest cluster in this subspace should be identified and the missing value replace by the value of the attribute at the center of this cluster. In effect localized averages are used to replace the missing values.

Treatment of outliers:

In some applications the training vectors may be mislabeled. This effect may be included by assigning probabilities of classes $p(C_i|R)$ which are not necessarily 0, 1, and optimizing the values of these probabilities. This should lead to base rate errors in regions where a few outliers survive. One method is to start from the initial labels, i.e. class probability $p(C_i|R) = \delta(C_i, C(R))$ and modify these probabilities to account for the neighborhood, adding just one parameter to preserve normalization:

$$p(C_i|R) \leftarrow (1-\gamma) p(C_i|R) + \gamma p(C_i|R;M) \tag{28}$$

This parameter should now be optimized.

Multi-model adaptive systems.

Instead of a single model that tries to provide one distance function in the whole input space several local distance functions are 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 orientation. Various procedures to combine results of different models may be defined, the simplest based on selection of the submodel with the minimum distance from the vector given for classification, and 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.

Relation to neural networks models.

Threshold neurons compute distances in a natural way. If the input signals $\{X_i = \pm 1\}$ and the weights $\{W_i = \pm 1\}$ are binary, neuron with N inputs and the threshold θ realizes the following function:

$$\Theta(\sum_i^N W_i X_i - \theta) = \begin{cases} 0 & \text{if } ||\mathbf{W}-\mathbf{X}|| > \theta \\ 1 & \text{if } ||\mathbf{W}-\mathbf{X}|| \leq \theta \end{cases} \tag{29}$$

where $||\cdot||$ norm is defined by the Hamming distance. One can interpret the weights of neurons in the first hidden layer as addresses of the reference vectors in the input space. Threshold neurons are activated by inputs falling into a hard sphere of radius θ centered at $\mathbf{W}$. Changing binary into real values and threshold into sigmoidal neurons, for inputs and weights normalized to $||\mathbf{X}|| = ||\mathbf{W}|| = 1$, leads to a soft activation of neuron by input vectors that are close to $\mathbf{W}$ on the unit sphere. In general the output function of a neuron:

$$\sigma(\mathbf{W}\cdot\mathbf{X}) = \sigma\left(\frac{1}{2}(||\mathbf{W}||^2 + ||\mathbf{X}||^2 - ||\mathbf{W}-\mathbf{X}||^2)\right) = \sigma(d_0 - d(\mathbf{W},\mathbf{X})) \tag{30}$$

For normalized input vectors sigmoidal functions (or any other monotonically growing transfer functions) simply evaluate the influence of reference vectors $\mathbf{W}$, depending on their distance $d(\mathbf{W},\mathbf{X})$, on classification probability $p(C_i|\mathbf{X};\{\mathbf{W},\theta\})$. As a function of a distance $\sigma(d_0 - d(\mathbf{W},\mathbf{X}))$ monotonically decreases, at $d(\mathbf{W},\mathbf{X}) = d_0$ reaching the value of 0.5. For normalized weights and inputs $\sigma(1-d(\mathbf{W},\mathbf{X})) \in [0.5, \sigma(1)]$, with $0 \leq d(\mathbf{W},\mathbf{X})) \leq 1$, i.e. only a part of the range of the sigmoidal function is used. For normalized $\mathbf{X}$ but arbitrary $\mathbf{W}$ the range of sigmoid argument is in the $[-|\mathbf{W}|, +|\mathbf{W}|]$ interval. Unipolar sigmoid has a maximum curvature around ± 2.4, therefore smaller weights of the norm mean that the network operates in almost linear regime. Regularization methods add penalty terms to the error function force the weights to become small and thus smoothing the network approximation to the training data.

Sigmoidal functions used in MLPs estimate the influence of weight vectors according to the distance between weight and the training vectors, combining many such estimations to compute the final output. RBF networks use Euclidean distance functions $d(\mathbf{X},\mathbf{R}^n) = ||\mathbf{X}-\mathbf{R}^n||$ and exponential $\exp(-d^2)$ weighting functions. Using combinations of sigmoidal functions (Eq. 24) to compute similarity of vectors has natural realization in form of neural network.

Interpretation of neural classifiers as a special minimal distance adaptive systems is thus feasible and worth of investigation. Neural networks used so far are based exclusively on Euclidean distance function. Using $\sigma(d_0 - d(\mathbf{W},\mathbf{X}))$ activation functions with different distance functions leads to new, more flexible neural networks with additional non-linear parameters. Big advantage of neural approaches is the ability to use fast gradient methods for error minimization. k-NN and similar distance-based methods require more expensive non-gradient minimization techniques.

III. Related work

A survey of the nearest neighbor methods has been published [23] but little effort has been spent on feature weighting. Wettschereck and Dietterich [25] 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. An alternative algorithm proposed here is:
Take all training reference vectors $\mathbf{R}$, look at k neighbors, store k for which $\mathbf{X}$ has correct class.
Classify $\mathbf{X}$ using the most common k, not the k which classifies correctly most of the closest M vectors to $\mathbf{X}$.

Lowe [22] introduced Variable Kernel Classifier based on Variable-kernel similarity Metric (VSM). In fact his approach is a version of RBF method because optimization of distance scaling factors for each feature is simply optimization of Gaussian dispersions. His formula for probability is:

$$p(C_i|X;M) = \frac{\sum_j^k G(X,R_j)p_i(R_j)}{\sum_j^k G(X,R_j)} \tag{31}$$

where $p_i(R_j)$ is the known probability that the reference vector R_j belongs to class C_i and $G()$ 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 unambigious (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 to select relevant inputs and assign them larger weights.

Very recently Yang et al. [27] introduced a constructive neural method “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 [9] 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 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 the 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 on a factor of 2 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. report very good results for many datasets the decision borders of such classifier are far from natural.

IV. Results

Results reported below are still preliminary. The nearest neighbor approach is frequently the best among many popular statistical, neural and machine learning classification methods. Methods presented in this paper should further improve these results.

k-NN – results reported here are based on increasing the number of neighbors until a tie is broken. Manhattan distance function was used in most studies ($\sum_i |X_i - Y_i|$). Both raw and standardized data (zero mean, unit variance) were used, since standardization does not always lead to improvement. We are using multistart simplex and adaptive simulated annealing [13] minimization methods to find the scaling factors for optimal metric. These methods are costly – minimization methods based on gradient techniques and simple search (“best-first" or “beam search") techniques should greatly improve the speed of convergence without significant decrease of accuracy.

Most of the data were taken from the UCI repository [14] of the Machine Learning Databases, where more detailed description may be found. Medical data include appendicitis (small data, 106 vectors, two classes, 7 attributes, data obtained from S. Weiss), Wisconsin breast cancer (699 cases, two classes, 9 attributes), Cleveland heart disease (303 cases, two classes, 13 attributes), hepatitis (155 vectors, two classes, 19 attributes) and a larger hypothyroid dataset described in more details below. Most medical datasets strongly benefited from normalization. The missing values of features have been replaced by the mean for their class, although this is not always the best procedure.

Detailed comparison of the scaled k-NN data with other approaches shows that even for small datasets results belong to the best reported so far. PVM, Cart, MLP and Bayes results are taken from [15], C-MLP2LN from [16], RIAC and C4.5 from [18] and FSM [19] are our own.

Unfortunately we have only a few results of the the leave-one-out tests to compare with. k-NN result for appendicitis is very good. Standardization did not improve the results although changed the optimal value of k from 8 or 9 to 23-25 (identical results for 3 values). Results from 10-fold crossvalidation are very similar for this dataset. Best results are obtained from feature selection: only 2 features, 3 and 5 were sufficient, other features either decrease or did not change accuracy. Adding 3 features that did not change accuracy and using scaling slightly better result was obtained.

For the Cleveland heart disease data the best results were obtained by FSM, with results from 10-fold crossvalidation being worse as much as 1.4%, but scaled k-NN results are also quite good. The best 10-fold crossvalidation results from DistAI give 85.3%. Feature scaling does not give significant improvement for this data set.

For the Wisconsin breast cancer data the best results were obtained by our FSM method [19], with results from 10-fold crossvalidation being worse by as much as 1.4%. Scaled k-NN results were not significantly different. DistAI results from the 10-fold crossvalidation obtained with the value difference metric gave 97.8%. This result and the FSM result show that global parameters may for this dataset be not sufficient for accurate classification.

For the hepatitis data scaled results with k=21 for the standardized data are excellent. Results for the Cleveland heart disease data are also quite good, although much better result obtained by FSM raises the question whether a global minimum for scaling parameters has really been obtained. The stratified 10-fold crossvalidation were almost identical to the leave-one out results, therefore it is interesting to give also DistAI results: the best results with standardized Euclidean metric were around 85% but since the deviation was almost 10% we suspect that the authors did not use stratified crossvalidation.

TABLE I

The appendicitis data. Accuracies given in % for the leave-one-out test

Method	Accuracy
Bayes rule (statistical)	83.0
CART, C4.5 (decision trees)	84.9
MLP+backpropagation	85.8
RIAC (prob. inductive)	86.9
25-NN, standardized, Manhattan	88.7
PVM, C-MLP2LN (logical rules)	89.6
8-NN, Manhattan, features 3 & 5	90.6
scaled 24-NN, Manhattan	90.6
scaled 21-NN, Manhattan, 5 features	91.5

TABLE II

The Cleveland heart disease data. Accuracies given in % for the leave-one-out test

Method	Accuracy
CART (decision tree)	80.8
MLP+backprop	81.3
FSM	82.2
C-MLP2LN	82.5
LDA	84.5
24-NN, Manhattan	84.8
26-NN, Manhattan, 1 feature removed	86.8

TABLE III

The Wisconsin breast cancer data. Accuracies given in % for the leave-one-out test

Method	Accuracy
RIAC	95.0
C4.5 (dec. tree)	96.0
3-NN, Manhattan	97.1
3-NN, Manhattan, 3 features removed	97.4
5-NN, as above, scaled	97.6
FSM	98.3

TABLE IV

The hepatitis data. Accuracies given in % for the leave-one-out test

Method	Accuracy
MLP+backprop	82.1
LDA	86.4
CART	82.7
FSM	90.0
21-NN	90.3
Scaled 3-NN	92.9

TABLE V

Classification results for the ann-thyroid dataset. Only the best MLP results are shown here.

Method	Training set accuracy %	Test set accuracy %
k-NN, Manhattan	–	93.8
Bayes rule	97.03	96.06
BP+conjugate gradient	94.6	93.8
Best Backpropagation	99.1	97.6
RPROP	99.6	98.0
k-NN, Manhattan, 5 features	–	98.0
Quickprop	99.6	98.3
BP+ genetic optimization	99.4	98.4
Local adaptation rates	99.6	98.5
Cascade correlation	100.0	98.5
PVM [15]	99.79	99.33
CART [15]	99.79	99.36
C-MLP2LN	99.89	99.36

The hypothyroid dataset was created from real medical tests screening for hypothyroid problems [14]. Since most people were healthy 92.7% of test cases belong to the normal group, and 7.3% of cases belonging to the primary hypothyroid or compensated hypothyroid group. A total of 3772 cases are given for training (results from one year) and 3428 cases for testing (results from the next year). Many neural classifiers have been tried on this data, giving accuracies up to 98.5% on the test set. Schiffman et.al. [17] optimized about 15 MLPs trained with different variants of backpropagation and cascade correlation algorithms and performed tedious genetic optimizations on many MLP network architectures. The best results of this study [17] reach 98.5% accuracy on the test set, while logical rules derived using MLP2LN method reach 99.1% accuracy.

The nearest neighbor method with Euclidean distance and raw data gives only 92.0% accuracy, below the base rate! Using the Manhattan distance on standardized data accuracy of 93.8% is obtained, still quite poor. The best *k*-NN results are obtained after selection of features. Features are turned off consecutively and the accuracy is checked on the standardized training set with the leave-one-out procedure. Those features that do not decrease accuracy were dropped. In effect 5 features were left (3 and 8, logical, and 13, 17, 21 continuos) and accuracy is increased to 98.0%. Although the number of reference vectors seems to be large one should remember that only 284 training and 250 test vectors from the hypo and hyperthyroid are present among the large number of data vectors. We have tried to use soft weighting with the conical function and optimized radius but the results were not significantly better (98.1% accuracy on the test set is obtained for k=5 and *r*=0.4, optimized using the leave-one-out procedure achieving 98.9% on the training set). Removing reference vectors, performing selection of features and optimizing the distance function using the sigmoidal functions should recover the rule-based solution giving 99.36% accuracy but it seems to be difficult to achieve.

Non-medical datasets included the ionosphere (200 cases, 2 classes), satimage (4435 cases, 6 classes), sonar (104 cases, 2 classes) and vovel (528 cases, 11 classes) data from the Machine Learning repository [14]. Only sonar data benefited from standardization. For the ionosphere we have the RIAC accuracy of 94.6% and C4.5 of 94.9% which are significantly worse than our results.

For satimage slightly better results (90.6% on the test) were obtained from conical soft-weighting of the influence of reference vectors, and for vowel the *r*-NN method gives 57.8% accuracy, but in both cases scaling improves the accuracy even more (of course one can combine scaling with soft-weighting and *r*-NN). We have looked at the correlation between the accuracy on the training and the test set using the conical weighting function $G(d) = 1 - d/\alpha r_k$, depending on the value of α. Peaks of maximum accuracy on the training data are not always correlated with peaks of maximum accuracy on the test data, but selecting the highest stable region (i.e. not a single peak but high plateau) on the training set (leave-one-out test) always gives significantly better result on the test set.

We have also tried the artificial data using the three monk problems. For the scaled 3-NN method in the first Monk problem 100% accuracy was obtained, for the second problem 85.9% (in this case MLP obtains 100%) and for the third problem 97.7%, which is an optimal result (the error is due to noise added to the data). All these result are still preliminary since we are developing the software that should implement many models described in this paper.

V. Summary and discussion

General conclusions that one may draw from these preliminary results are: 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-optimized soft weighting is performed.

The minimal distance point of view leads to a fruitful framework in which many methods are accommodated. We have found very few methods in the literature that try to improve upon the simple *k*-NN scheme. Hastie and Tibshirani [20] 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 [21] proposed an interesting way of adapting the metric based on a tree-structure interactive partitioning of the data. Laaksonen and Oja [6] proposed to improve the *k*-NN reference vectors using LVQ techniques. Atkenson, Moor and Schaal [12] discuss locally weighted regression techniques, minimal distance methods with various metric and kernel functions applied to approximation problems.

All these proposals and many more 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 *k*-NN as well as neural classifiers. Many possibilities to create fuzzy *k*-NN models remain to be explored. 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. Bearing in mind that so far we have tested only a few simplest methods our results, even for small datasets, are very encouraging.

Acknowledgments: Support of W.D. and K.G. by the Polish Committee for Scientific Research is gratefully acknowledged.

References

[1] W. Duch, Neural minimal distance methods, Proc. 3-rd Conf. on Neural Networks and Their Applications, Kule, Poland, Oct. 14-18, 1997

[2] D. Michie, D.J. Spiegelhalter and C.C. Taylor, Machine learning, neural and statistical classification. Elis Horwood, London 1994

[3] R. Rohwer and M. Morciniec, A Theoretical and Experimental Account of n-tuple Classifier Performance, *Neural Computation* 8 (1996) 657–670

[4] P.R. Krishnaiah, L.N. Kanal, eds, Handbook of statistics 2: classification, pattern recognition and reduction of dimensionality (North Holland, Amsterdam 1982)

[5] T.M. Mitchell, Machine Learning. McGraw-Hill 1997

[6] J. Laaksonen, E. Oja, Classification with Learning *k*-Nearest Neighbors. In: *Proc. of ICNN'96*, Washington, D.C., June 1996, pp. 1480-1483.

[7] D.L. Waltz, Memory-based reasoning, in: M. A. Arbib, ed, *The Handbook of Brain Theory and Neural Networks* (MIT Press 1995), pp. 568–570; C. Stanfill and D. Waltz, *Toward memory-based reasoning*, Communications of ACM 29 (1986) 1213-1228

[8] R.P. Lippmann, An introduction to computing with neural nets, *IEEE Magazine on Acoustics, Signal and Speech Processing* 4 (1987) 4–22; P. Floreen, The convergence of Hamming memory networks, *Trans. Neural Networks* 2 (1991) 449–457

[9] D.L. Reilly, L.N. Cooper, C. Elbaum, A neural model for category learning, *Biological Cybernetics* 45 (1982) 35–41

[10] P.D. Wasserman, Advanced methods in neural networks (van Nostrand Reinhold 1993)

[11] D.K.Y. Chiu, F.E. Kavanaugh, the ck-Nearest Neighbor distance Network: a network using class boundary feature distances, ICONIP'97, New Zealand, Nov.1997, pp. 535-538

[12] C.G. Atkenson, A.W. Moor and S. Schaal, Locally weighted learning, *Artificial Intelligence Review* (submitted, 1997)

[13] L. Ingberg, *Adaptive simulated annealing (ASA): Lessons learned*, J. Control and Cybernetics 25 (1996) 33-54

[14] C.J. Mertz, P.M. Murphy, UCI repository, http://www.ics.uci.edu/pub/machine-learning-databases.

[15] S.M. Weiss, I. Kapouleas, An empirical comparison of pattern recognition, neural nets and machine learning classification methods, in: J.W. Shavlik and T.G. Dietterich, *Readings in Machine Learning*, Morgan Kauffman Publ, CA 1990

[16] W. Duch, R. Adamczak, K. Grąbczewski, Extraction of crisp logical rules using constrained backpropagation networks. *ICANN'97*, Houston, 9-12.6.1997, pp. 2384-2389, Logical rules for classification of medical data using ontogenic neural algorithm, *EANN'97*, Stockholm, 16-18.06.1997, pp. 199-202

[17] W. Schiffmann, M. Joost, R. Werner, Comparison of optimized backpropagation algorithms, *ESANN '93*, Brussels 1993, pp. 97-104; Synthesis and Performance Analysis of Multilayer Neural Network Architectures, Tech. Rep. 15/1992, available in `neuroprose` as schiff.gann.ps.Z

[18] H.J. Hamilton, N. Shan, N. Cercone, RIAC: a rule induction algorithm based on approximate classification, Tech. Rep. CS 96-06, Regina University 1996

[19] W. Duch, G.H.F. Diercksen, Feature Space Mapping as a universal adaptive system, *Comp. Phys. Comm.* 87 (1995) 341-371

[20] T. Hastie, R. Tibshirani, Discriminant adaptive nearest neighbor classification, *IEEE PAMI* 18 (1996) 607-616

[21] J. H. Friedman, Flexible metric nearest neighbor classification, *Technical Report, Dept. of Statistics, Stanford University* 1994

[22] D.G. Lowe, Similarity metric learning for variable-kernel classifier, *Neural Comp.* 7 (1995) 72-85

[23] B.V. Dasarathy, ed, Nearest neighbor norms: NN Pattern Classification Techniques, IEEE Computer Society Press, New York 1991

[24] D. Wettschereck, D.W. Aha, and T. Mohri, A Review and Empirical Evaluation of Feature Weighting Methods for a Class of Lazy Learning Algorithms, Artificial Intelligence Review 11 (1997) 273-314

[25] D. Wettschereck, T. G. Dietterich, Locally adaptive nearest neighbor algorithms, NIPS 6 (1994) 184-191

[26] B.W. Silverman, Density estimation for statistics and data analysis, Chapman and Hall, London 1986

[27] J. Yang, R. Parekh, V. Honavar, DistAI: an inter-pattern distance-based constructive learning algorithm, World Congress on Computational Intelligence, Anchorage, Alaska, May 1998, pp. 2208-2213