THE WEIGHTED *k*–NN WITH SELECTION OF FEATURES AND ITS NEURAL REALIZATION.

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

Abstract

As a step towards neural realization of various similarity based algorithms *k*-NN method has been extended to weighted nearest neighbor scheme. Experiments show that for some datasets significant improvements are obtained. As an alternative to the minimization procedures a best–first search weighted nearest neighbor scheme has been implemented. A feature selection method for *k*-NN, based on a variant of the best–first search strategy, has also been implemented. This method is relatively fast and for some databases gives excellent results. Finally a natural neural network extension of *k*-NN method is described, including weights and other parameters as a part of the model.

I. INTRODUCTION

RECENTLY a general framework for similarity-based methods has been presented [1]. Except for classical minimal distance methods, such as *k*-NN, many popular neural network models (such as MLP, RBF, LVQ or SOM models) may be presented in this form [2]. Some of the simplest and frequently the most accurate classification algorithms applicable to pattern recognition problems are based on the *k*-nearest neighbor (*k*-NN) rule [3]. 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 [4]. 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 $k = 1$ and 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 [3]. In practice the number of reference vectors is relatively small and small values of k work better.

Because the *k*-NN method is so simple it should be used as a standard reference for other classificators. Unfortunately and surprisingly very few computer programs are around even for the simplest version of *k*-NN. A possible practical problem for large datasets is the computational complexity of the actual classification step, demanding for n reference vectors calculation of $\sim n^2$ distances and finding k smallest distances. Although Laaksonen and Oja [5] 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. 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.

In similarity-based methods, such as *k*-NN, parametrization of distance measures is of great importance [1]. Frequently reducing the number of features and assigning scaling factors to features leads to significant improvements. In the next two sections algorithms for feature selection and scaling are presented. Only one neural model proposed so far is explicitly based on the nearest neighbor rule: the Hamming network [6] computes the Hamming distances for the binary patterns and finds the maximum overlap (minimum distance) with the prototype vectors, realizing the 1-NN rule. In the fifth section another neural realization, which is a significant extension of the *k*-NN, is presented. A short summary is presented in the last section.

II. SELECTION OF FEATURES

IT is known that instance (memory) based algorithms degrade in performance (prediction accuracy) when faced with many features that are not necessary for predicting the desired output. After experimentation with various feature selection methods we have developed a method based on a variant of "the best first" search strategy. This method is relatively fast and applied to the *k*-NN method for some databases gives excellent results. It is worth to mention that one of the authors (KG) was once converting a diabetes dataset [7] to the format of *k*-NN package using AWK utility. Because of the error in the script as a second feature a class number was added. Our selection of features method immediately picked out this feature as the most important rejecting all other features and classification accuracy of 100% was correctly reported. This simple test should be used for all feature selection methods.

The **feature dropping** algorithm proceeds as follows: features are turned off one after another, one at a time. The leave-one-out test is performed on the training file and results improve when some (unwanted) features are turned off and degrade when some (important) features are turned off. Feature leading to the highest classification accuracy improvement on the training file is selected as the least important. This feature is removed from the input set and the whole procedure repeated until only one feature is left in the input set, presumably the most important one.

Although this algorithm may be used with any method is is particularly easy to use with the *k*-NN because the leave-one-out test is very easy to perform. Apart from the value of k there is no learning and the cost of each step is equal to the cost of classification of all training vectors. If the cross-validation calculations are performed the leave-one-out test is done on the training partition of the dataset. The number of the leave-one-out evaluations in the worst case is $N \cdot (N+1)/2 - 1$.

The artificial example presented below illustrates the feature dropping algorithm for a dataset containing five attributes. Features that are turned off at each step are listed and the leave-one-out test on the training file is performed.

Level 1: f1, f2, f3, f4, f5; best improvement for f4

Level 2: f4 f1, f4 f2, f4 f3, f4 f5; best improvement for f5

Level 3: (f4 f5) f1, (f4 f5) f2, (f4 f5) f3; best improvement for f1

Level 4: (f4 f5 f1) f2, (f4 f5 f1) f3

After all $N-1$ levels are performed the sequence of features that should be removed to achieve the highest classification accuracy on a training file is selected. It may happen that at some level there is no improvement and the feature which leads to minimal degradation is selected. If the number of features is large and there is no improvement for several levels the procedure may be stopped, because there is little chance that results will improve at a later stage. Standardization of the input vectors before the feature selection procedure is usually desired. The method is fast if the number of features is not to high but considering the datasets such as DNA with 180 attributes the selection of features requires over 16000 leave-one-out tests. A partial ranking of features may be done at a lower cost. The least expensive method, requiring N tests, stops after the first level. If all feature were completely independent and the effects of feature removal were additive this would be sufficient. The full $N-1$ level selection procedure may be performed using the subset of features identified as promising at the first level (i.e. those features that may be removed without degradation of the accuracy of k-NN). Other search strategies may be used to select the best combination of features [8]. The feature dropping method may be efficiently parallelized to reduce the time of calculations.

III. SCALING FACTORS IN SIMILARITY MEASURES

FEATURE selection assigns binary scaling factors to each feature: $s_i = 0$ or 1. Minkovsky's distance with the scaling factors is:

$$D(\mathbf{A},\mathbf{B};s)^{\alpha} = \sum_{i}^{N} s_i d(\mathbf{A}_i,\mathbf{B}_i)^{\alpha} \tag{1}$$

The scaling factors s_i determine relative importance of different features. In this section two groups of methods that automatically assign such scaling factors are proposed. The first group of methods is based on minimization of the cost function and the second group on the best-first search applied to discretized scaling factors. Although we have used k-NN proposed methods of scaling factor determination may be used with any similarity-based classifier.

Methods based on minimization. The cost function is simply the number of classification errors the classifier makes. We use three minimization procedures: simplex method (a local method), adaptive simulated annealing (ASA, a global method) and multisimplex global minimization method. The local simplex method usually requires less than 100 evaluations and is the fastest but results obtained with this method have large variance. It is a good method to start from to see whether the optimization of features works for a particular database. For a dataset having separate training file ASA or multisimplex method converge to similar results in all calculations (the advantage of global minimization). However, global optimization methods are expensive and may require a large number of accuracy evaluations for convergence.

Best first search methods. We will discuss three methods belonging to this group. They are designated as $S0$, $S1$ and the scale-tuning method. The last algorithm is used to search for optimal scaling factors starting from a solution obtained by other methods. Our experiments indicate that search-based algorithms are faster and usually give better accuracy than minimization methods. Since scaling factors are real-valued they have to be quantized, either with fixed precision (in method $S0$ and $S1$) or precision that is steadily increased during the progress of the search procedure (scale-tuning method).

The $S0$ algorithm starts from a selection of a single most important feature. Classification method is used with just one feature, ranking the performance of each features. Features that lead to highest performance are good candidates to start from, and their indices are stored in an array in decreasing rank order. The scaling factor s_1 corresponding to feature which has highest rank is set as 1, defining the relative distance scale. The scaling factor corresponding to the next-highest ranked feature s_2 is determined by evaluating the classifier's performance for two input features, with $s_2 = m\Delta \in [0,1]$ (the default is $\Delta = 0.05$). The best s_2 and classification accuracy is stored (at present if there are several scaling factors leading to the same highest accuracy – only the first is remembered and the rest is ignored). The process is repeated taking the next-highest ranking feature $s_3 = m\Delta \in [0,1]$ with s_1, s_2 fixed, and evaluating the performance for 3 features. Finally after all scaling factors are determined the performance of the classifier on the test set with the scaled similarity function is evaluated. The total number of evaluations on the training set is on the order of N/Δ. The method corresponds to quantized version of the line search optimization procedure.

Unfortunately the $S0$ algorithm frequently does not work for datasets for which the selection of features works. Better results are usually obtained by dropping, rather than adding features. The $S1$ algorithm is 'a mirror image' of the $S0$ algorithm. In the initial ranking of the features all scaling factors are set to 1 and evaluation with a single feature turned off $s_i = 0$ is made for $i = 1 \ldots N$. Thus the ranking is done in the same way as in the feature dropping selection method. The most important feature has a fixed value of the scaling factor $s_1 = 1$ and the optimal scaling factor for the second feature in the ranking is determined by the search procedure, while the remaining scaling factors are all fixed to 1. The search is repeated for feature that has next-highest rating, with optimal values of scaling factors for higher ranked features kept fixed and lower-ranked features left at 1, until all scaling factors are determined.

The last search-based method is used to tune the scaling factors already found by some other procedure (either by a minimization method or one of the methods described above). In this method scaling factors are changed without initial ranking, from first to the last one. Scales are changed for every feature by adding or subtracting a constant value $s_i \leftarrow s_i \pm \delta$ where δ is a parameter given by the user (by default $\delta = 0.5$). If change of the scaling factor leads to an improvement of the classification accuracy the scaling factor is updated, otherwise it remains unchanged. After the last scaling factor is checked in this way the δ parameter is divided by 2 and the whole procedure repeated. The algorithm is terminated if the difference in classification accuracy during two subsequent iterations is smaller than a given threshold.

IV. RESULTS

DUE to the lack of space only a few results are presented. In the well known hypothyroid dataset [7] despite the high number of training cases (3772) the k-NN method (after selection of k and selection of the distance measure) performs only slightly better than the majority classifier, giving classification accuracy of 94.4% on the test set (3428 cases). This is much worse than most other algorithms, including neural networks and logical rules [9]. After applying the feature dropping algorithm from 21 features present in the original dataset only 4 remain (f3, f8, f17, f21), increasing the classification accuracy on the test set by nearly 3% to 97.9% (a significant improvement since the number of cases in the test set is large). Applying scaling method $S1$ and tuning the scaling factors we were able to increase the classification accuracy further to 98.1%. This is probably the best result for this database obtained so far with the minimal distance method although still significantly worse than the result obtained with logical rules.

Another interesting database is the hepatobiliary disorders data obtained from Tokyo Dental and Medical University [9] (536 cases, including 163 test cases, 9 features, 4 classes).

k-NN with Manhattan distance function gives 77.9% accuracy for this dataset which is already much better than other methods [10] (for example MLP trained with RPROP gives accuracies that are below 70%). After applying feature selection method 4 features were removed (features 2, 5, 6 and 9), increasing accuracy to 79.1%. Using scaled *k*-NN methods it was possible to increase the accuracy further to 82.8%. These results are significantly better than for all other classifiers applied to this data (including IB2-IB4, FOIL, LDA, DLVQ, C4.5, FSM, Fuzzy MLP and K* methods).

We have also tried the 3 artificial datasets for the Monk problems, popular in machine learning community [7]. For the Monk1 problem (124 train, 432 test cases, 2 classes, 6 features) *k*-NN with Euclidean metric, $k = 1$, gives 89.5% accuracy on the test set, while minimization of the scaling factors with the simplex method increases accuracy to 97.2%, and with multisimplex even to 100%. The best-first search scaling method or feature selection gives also 100% accuracy. For the Monk2 problem (169 train, 432 test cases, 2 classes, 6 features) *k*-NN gives 82.6% which is increased by scaling to 84.5%, still rather disappointing since many rule-based methods may find the original classification rules achieving 100% accuracy and MLPs are also capable of such accuracy. Perhaps this is a difficult problem for the memory-based methods such as *k*-NN. Many other methods achieve less than 80% of accuracy for this problem [7]. For the Monk3 problem (122 train, 432 test, 2 classes, 6 features) *k*-NN gives 89.1% accuracy and is significantly improved by using the simplex minimization (93.3%), multisimplex (92.4%) and ASA (94.2%). The best results, 97.2% accuracy on the training set, was obtained using best-first feature scaling followed by tuning the scaling factors.

V. NEURAL REALIZATION OF THE MINIMAL-DISTANCE METHOD

THE network realization described here is a generalization of the *k*-NN method and should improve the results further. The network has hidden nodes computing distances $D(\mathbf{X}-\mathbf{R})$, where $\mathbf{R}$ are reference (training) vectors. k nodes with the smallest distances output their class label $h_j(\mathbf{X};\mathbf{R}) = C_i$ and the remaining nodes output $h_j(\mathbf{X};\mathbf{R}) = 0$. The classes are numbered from $C_i = 1 \ldots N_C$. The output layer computes probabilities using the formula:

$$P(C_i|\mathbf{X};M) = \sum_j \mathbf{W}_{ij} \cdot h_j(\mathbf{X}) \qquad p(C_i|\mathbf{X};M) = \frac{P(C_i|\mathbf{X};M)}{\sum_i P(C_i|\mathbf{X};M)} \tag{2}$$

The weights W_{ij} between output node computing probabilities for class C_i are equal to $W_{ij} = \mathbf{S}(C_i,C_j)/C_j$, where the matrix $\mathbf{S}(\cdot)$ estimates similarity among the output classes and in the traditional *k*-NN is replaced by a Kronecker 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 C_j class, contributes to the probability of the C_i class a value $S(C_i,C_j)$. The structure of the network is shown in Fig. 1. For the cost function that should be optimized one may take:

$$E(\mathbf{W},k) = \sum_{\mathbf{X}} \sum_i R\left(C_i,C(\mathbf{X})\right)\left(p(C_i|\mathbf{X};M) - \delta(C_i,C(\mathbf{X}))^2\right) \tag{3}$$

where the model M includes k as parameter and the $S(C_i,C_j)$ is the output-class similarity matrix. If we want to minimize the number of classification errors output probabilities should be changed into binary 0, 1 values by the winner-takes-all procedure.

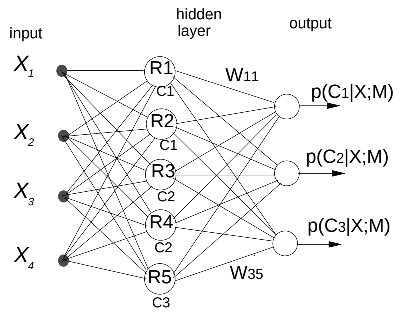

Fig. 1. Network generalization of the *k*-NN method. The hidden nodes compute distances to reference vectors and return *k* values of class labels associated with the nodes, while the output nodes compute probabilities.

The output weights, initialized to $W_{ij} = \mathbf{S}(C_i, C_j)/C_j$, may be treated as adaptive parameters. Introduction of soft weighting $G(D(\cdot))$ allows to use gradient optimization methods. For many datasets (especially for images) this simple network may outperform MLPs and other neural models, since the results should be at least as good as the *k*-NN results.

VI. SUMMARY

SURPRISINGLY few methods in the literature try to improve upon the simple *k*-NN scheme. In this paper several aspects of the similarity based methods were discussed: selection of features and determination of the feature scaling factors. Several new methods were introduced, significantly improving the results obtained by the straightforward *k*-NN approach. A natural network realization of *k*-NN method leads to a model with more parameters and should allow to improve the results even further. It is applicable to problems with an infinite number of output classes and using the matrix $\mathbf{S}(\cdot)$ may take into account the similarity of the output classes. We are convinced that investigation of connections between neural networks and similarity based methods is a fruitful task.

Acknowledgment: We would like to thank the Polish Committee for Scientific Research, grant no. 8T11F 014 14, for partial support of this work.

References

[1] W. Duch, Neural minimal distance methods, Proc. 3-rd Conf. on Neural Networks and Their Applications, Kule, Poland, Oct. 14-18, 1997; W. Duch, K. Grudziński, *A framework for similarity-based methods.* 2nd Polish Conf. on Theory and Applications of Artificial Intelligence, Łódź 1998, pp. 33-60

[2] W. Duch, K. Grudziński, G.H.F. Diercksen, *Minimal distance neural methods.* Proc. IJCNN'98, pp. 1299-1304; W. Duch, R. Adamczak, G.H.F. Diercksen, *Distance-based multilayer perceptrons.* CIMCA, Ed. M. Mohammadian, IOS Press, Amsterdam, pp. 75-80

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

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

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

[6] P. Floreen, The convergence of Hamming memory networks, *Trans. Neural Networks* 2 (1991) 449–457

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

[8] L. Kanal, V. Kumar (Eds), Search in Artificial Intelligence (Springer Verlag 1988)

[9] W. Duch, R. Adamczak, K. Grąbczewski, G. Żal, *Hybrid neural-global minimization method of logical rule extraction.* Journal of Advanced Computational Intelligence (in print)

[10] W. Duch, R. Adamczak, K. Grąbczewski, G. Żal, Y. Hayashi, *Fuzzy and crisp logical rule extraction methods in application to medical data.* Fuzzy Systems in Medicine, Springer 1999 (in print)