Information Sciences 179 (2009) 2964–2973

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

A method of learning weighted similarity function to improve the performance of nearest neighbor

Mansoor Zolghadri Jahromi [a,*,1], Elham Parvinnia [b], Robert John [a]

[a] Centre for Computational Intelligence, Department of Informatics, De Montfort University, Leicester, LE1 9BH, UK
[b] Department of Computer Science and Engineering, Shiraz University, Shiraz, Iran

ARTICLE INFO

Article history:
Received 17 November 2008
Received in revised form 17 March 2009
Accepted 20 April 2009

Keywords:
Nearest neighbor
Weighted metrics
Adaptive distance measure
Prototype reduction

ABSTRACT

The performance of Nearest Neighbor (NN) classifier is known to be sensitive to the distance (or similarity) function used in classifying a test instance. Another major disadvantage of NN is that it uses all training instances in the generalization phase. This can cause slow execution speed and high storage requirement when dealing with large datasets. In the past research, many solutions have been proposed to handle one or both of the above problems. In the scheme proposed in this paper, we tackle both of these problems by assigning a weight to each training instance. The weight of a training instance is used in the generalization phase to calculate the distance (or similarity) of a query pattern to that instance. The basic NN classifier can be viewed as a special case of this scheme that treats all instances equally (by assigning equal weight to all training instances). Using this form of weighted similarity measure, we propose a learning algorithm that attempts to maximize the leave-one-out (LV1) classification rate of the NN rule by adjusting the weights of the training instances. At the same time, the algorithm reduces the size of the training set and can be viewed as a powerful instance reduction technique. An instance having zero weight is not used in the generalization phase and can be virtually removed from the training set. We show that our scheme has comparable or better performance than some recent methods proposed in the literature for the task of learning the distance function and/or prototype reduction.

1. Introduction

The NN rule is one of the oldest and simplest methods of non-parametric pattern classification. The basic rationale for the NN rule is both simple and intuitive: patterns close in feature space are likely to belong to the same class. The NN classifier can be represented by the following simple rule: to classify an unknown pattern, choose the class of the nearest stored training instance. A common extension is to choose the most common class among the K Nearest Neighbors (KNN).

Despite its simplicity, the NN classifier has many advantages over other methods. For example, it can learn from a small set of examples, it can incrementally add new examples as they become available, and it can give competitive performance with other methods such as decision trees or neural nets. However, it suffers from the following problems:

* Corresponding author. Tel.: +44 7948917982.
E-mail addresses: zjahromi@shirazu.ac.ir (M.Z. Jahromi), parvinn@shirazu.ac.ir (E. Parvinnia), rij@dmu.ac.uk (R. John).
[1] On sabbatical leave from Shiraz University.

0020-0255/$ - see front matter © 2009 Published by Elsevier Inc.
doi:10.1016/j.ins.2009.04.012

1. The performance of the NN rule depends crucially on the distance metric used to find the NN of a query pattern.
2. The basic NN classifier stores all of the training instances to use during the generalization phase. To classify an input test pattern, its distance to all stored instances must be calculated. This can cause slow execution speed when dealing with large datasets.

To tackle the first problem, many methods have been developed to locally adapt the distance metric. Examples of these include the flexible metric method proposed by Friedman [6], the discriminate adaptive method by Hasti and Tibshirani [7], and the adaptive metric method by Domeniconi et al. [4]. The common idea underlying the above methods is that they estimate feature relevance locally at each query pattern. This leads to a weighted metric for computing the similarity between the query pattern and training data. For example, in [16], an adaptive KNN classification algorithm is proposed that is based on the concept of statistical confidence from hypotheses testing.

In [15], a locally adaptive distance measure is used that is based on assigning a weight to each training instance. However, the parameters of the distance measure (i.e., the weights of the training instances) are specified by a simple heuristic. This scheme is shown to be effective in improving the performance of the basic NN.

In [12], a scheme is proposed to learn weighted metrics to improve generalization accuracy of the basic NN. The weights (i.e., the parameters of the distance measure) may be specified for each class, feature, or individual instance. The learning algorithms uses gradient descent to minimize a performance index that is an approximation to LV1 classification error-rate of the given training set.

Another work in this field is the scheme presented in [5] to learn the distance function for NN rule. The distance function is learned based on maximizing the clustering of objects belonging to the same class. Objects belonging to a dataset are clustered with respect to a given distance function and the local class density information of each cluster is then used by a weight adjustment heuristic to modify the distance function so that the class density is increased in the attribute space.

Adaptation of the distance measure has been the subject of research in other distance-based classifiers. For example, in [20,21] schemes are proposed to adapt the distance measure by assigning a weight to each fuzzy rule.

By allowing all the input features to contribute to the distance metric equally, irrelevant, redundant, or noisy features can degrade classification accuracy of the nearest neighbor classifier. Various feature selection/weighting methods proposed in the literature [1,8,14] can be used to tackle this problem.

Similar to [15], the weighted metric we use in this paper is based on assigning a weight to each instance in the training set. However, the weights of the training instances are specified by the learning algorithm that we propose for this purpose (i.e., instead of the heuristic method used in [15]). Our proposed learning algorithm attempts to specify the weights of training instances such that LV1 classification error-rate of the given training set is minimized. We denote our method as Weighted Distance Nearest Neighbor (WDNN) in this paper.

To tackle the second problem mentioned above (i.e., improving the generalization speed and storage requirements of the basic NN classifier), instance reduction techniques [9,18] aim at reducing the size of training set. One other reason to reduce the original dataset is to improve the prediction accuracy of the basic NN classifier by removing noisy instances from the training set. The intuition behind these algorithms is that internal instances do not affect the decision boundary as much as border instances and therefore they can be removed to find a significantly reduced subset. Wilson and Martinez in [17] have provided a survey of the algorithms developed for this purpose. They have also proposed five instance reduction algorithms (DROP1-5) in [17,18].

In [19], the authors use the concept of clustering to reduce a dataset by a compact set of representatives. They propose a supervised clustering algorithm to find a set of representatives for a given classification task. In their scheme denoted as Supervised Clustering Editing/Nearest Representative (SCE/NR), a test instance is classified by finding its nearest representative (instead of using the full training set).

In [11], a prototype reduction algorithm is proposed that simultaneously searches for a reduced subset of prototypes and a suitable local metric for these prototypes. This algorithm starts with an initial subset of prototypes selected in random, and then iteratively adjusts both the position and the corresponding local-metric weights. The resulting prototypes/metric combination minimizes a suitable estimation of the classification error probability.

In [10], a method of prototype selection is proposed that assign a weight to each selected prototype. However, to specify the weight of a prototype, it uses a naïve weight assignment approach by simply searching through a set of predefined weight values.

In [13], genetic algorithm is used to simultaneously search for a reduced subset of instances and features. In facing problems with many irrelevant features, the scheme is shown to be quite effective in improving the performance of the basic nearest neighbor.

The proposed method prescribed in this paper (i.e., WDNN) tackles both problems of the NN classifier (mentioned earlier in this section). That is, it simultaneously searches for a reduced subset of prototypes and a suitable local metric for these prototypes. In this respect, it has the same goal as [11]. However, in our method, the prototypes are selected from the actual dataset examples while the method presented in [11] constructs artificial prototypes. Algorithms that use actual dataset examples as prototype are more applicable when data is represented by nominal attributes.

The rest of this paper is organized as follows. In Section 2, NN classification with weighted training instances is discussed to introduce the notation. In Section 3, our weight-learning methods (i.e., WDNN and MWDNN) are presented. In Section 4, simulation results are given. Section 5 concludes the paper.

2. Nearest neighbor classification with weighted instances

We briefly describe the NN rule to introduce the notation. For an M-class problem, assume that a set of training examples of the form $\{(X_i, C_i) \mid i = 1,\ldots,n\}$ is given. Where, $X_i = [x_{i1}, x_{i2}, \ldots, x_{id}]^T$ is a d-dimensional vector of attributes and $C_i = [1, 2, \ldots, M]$ defines the corresponding class label. To identify the NN of a query pattern, a distance function $d(X_i, X_j)$ should be defined to find the NN of a query pattern. A variety of distance functions have been proposed for this purpose [18].

Instead of working with the distance function, we can equivalently work with the following similarity measure, which normalizes the similarity value μ of any two instances (i.e., X_i and X_j) to a number in the interval [0,1].

$$\mu(X_i, X_j) = 1 - (d(X_i, X_j)/d_{\max}) \tag{1}$$

where, $d_{\max}$ is the maximum possible distance between two instances in the feature space, which is calculated by considering two virtual instances having maximum difference in each attribute value. That is:

$$d_{\max} = \sqrt{\sum_{i=1}^{d} (\Delta_i)^2} \tag{2}$$

where, Δ_i represents the difference between maximum and minimum values of the attribute i.

Using (1), the most similar pattern X_p to a query pattern Q can be formally stated as:

$$p = \arg\max_{1 \leqslant j \leqslant n} \{\mu(Q, X_j)\} \tag{3}$$

In this paper, we assign a weight w_k to each stored instance X_k. The weights of the stored instances are used in the test phase to find the most similar pattern X_w to a query pattern Q.

$$w = \arg\max_{1 \leqslant j \leqslant n} \{w_j \cdot \mu(Q, X_j)\} \tag{4}$$

3. Learning the weights of training instances

In this section, we present the WDNN algorithm that attempts to maximize the LV1 classification rate of the nearest neighbor classifier by assigning a weight (in the interval $[0,\infty]$) to each training instance. In fact, this algorithm is the modified version of the algorithm that was proposed in [21] to construct the rule-base of a fuzzy classification system.

For an M-class problem, assume that a training set $Tr = \{X_i \mid i = 1,\ldots,n\}$ consisting of n labeled patterns from different classes are available. Initially, the weights of all training instances are set to one $\{$i.e., $w_i = 1$, $i = 1,\ldots,n\}$.

In its basic form, the proposed WDNN algorithm is a hill-climbing search method. The algorithm starts with an initial solution to the problem (i.e., $\{w_i = 1, i = 1,\ldots,n\}$) and sequentially improves the solution by finding a "neighbor solution" that is better than the current one. The WDNN considers a solution as "neighbor" if it is different in the weight of just one instance with respect to the current solution. In order to find a neighbor solution that is better than the current one, the WDNN algorithm specifies the best weight w_k (i.e., resulting in maximum LV1 classification rate) of a typical instance X_k assuming that, the weights of other instances are given and fixed. In this way, the WDNN algorithm improves the current solution by visiting each instance and specifying its best weight. Obviously, when the weights of other instances change during the execution of the WDNN algorithm, the specified weight w_k is not optimal anymore. That is why the second and subsequent passes over the data can further improve the LV1 classification accuracy. In our implementation, we terminate the search after a fixed number of passes over the entire training set (i.e., the loop at line 9 of the WDNN algorithm).

The WDNN algorithm presented in Table 1 start by finding the *associates* of each training instance (the loop at line 5). *Associates* of a training instance X (denoted as $A(X)$) are those training instances that have X as their nearest neighbor in LV1 test. The WDNN algorithm keeps the associate list and the nearest neighbor of each training instance in memory and updates them as the weights of instances change during the execution of the algorithm.

The WDNN algorithm specifies the weight w_k of a typical instance $X_k \in ClassT$ as follows (the loop at line 10). In the first step, X_k is removed from the attribute space by setting its weight to zero (at line 11). This forces each of its associates to use a new nearest neighbor in LV1 test. Update of associate lists is performed by removing each associate of X_k from $A(X_k)$ and adding it to its new neighbor's list of associates. To specify the best weight of X_k, the WDNN algorithm needs to identify those training instances that their correct/incorrect LV1 classification depends on the value of w_k. To identify these instances, the WDNN algorithm marks two groups of instances that their correct/incorrect LV1 classification does not depend on the value of w_k. These are:

(1) Instances of *ClassT* that are classified correctly even if X_k is removed from the training set. These instances are classified correctly regardless of the value of w_k.
(2) Instances of *NOT-ClassT* that are misclassified even if X_k is removed from the training set. These instances are misclassified regardless of the value of w_k.

At this stage, correct/incorrect LV1 classification of unmarked instances depends on the value of w_k.

Table 1
The WDNN algorithm for finding the weights of training instances.

```
Input: training instances $Tr = \{X_i, i = 1,2,\ldots,n\}$
Output: weight array $w = \{w_i, i = 1,2,\ldots n\}$
1.  WDNN(Training set $Tr$): Weight array $w$
2.    For $i = 1$ to $n$
3.      Let $w_i = 1$
4.      Let $A(X_i) = \{\phi\}$
5.    For each instance $X_k$ in $Tr$
6.      Find the nearest neighbor of $X_k$ {assume that $X_l$ is the nearest neighbor of $X_k$}
7.      $NN(k) = l$ {$NN(k)$ is used to denote the nearest neighbor of $X_k$}
8.      Add $X_k$ to $A(X_l)$
9.    For $i = 1$ to no. of iterations
10.     For each instance $X_k$ in $Tr$ {Assume that $X_k \in ClassT$}
11.       Let $w_k = 0$ {i.e., remove the instance from the feature space}
          {updating the associate lists}
12.       For each instance $X_p$ in $A(X_k)$
13.         Find the new nearest neighbor of $X_p$ {assume that $X_l$ is the nearest neighbor of $X_p$}
14.         Remove $X_p$ from $A(X_k)$
15.         Add $X_p$ to $A(X_l)$
16.         $NN(p) = l$
17.       Mark instances in $Tr$ that are from $ClassT$ and classified correctly
18.       Mark instances in $Tr$ that are not from $ClassT$ and misclassified
19.       Use (5) to rank the unmarked instances in ascending order of their scores
20.       current = no. of instances in unmarked list that are classified correctly
21.       best = current
22.       best-th = 0
          {Assume that $X_t$, $X_{t+1}$ represent any two successive patterns in the ranked list}
23.       For each different threshold $th = (Score(X_t) + Score(X_{t+1}))/2$
24.         current = no. of instances that are classified correctly
25.         If current > best then
26.             best = current
27.             best-th = th
          {Assume that $\lambda$ is the score of the last pattern in the list and $\tau$ is a very small positive number}
28.       $th = \lambda + \tau$
29.       current = no. of instances in the list that are classified correctly
30.       If current > best then
31.         best = current
32.         best_th = th
33.       $w_k$ = best_th
          {updating the associate lists}
34.       For each instance $X_p$ in $Tr$ {assume that $X_r$ is the current nearest neighbor of $X_p$}
35.         If score($X_p$) < $w_k$
36.             Add $X_p$ to $A(X_k)$
37.             Remove $X_p$ from $A(X_r)$
38.             $NN(p) = k$
39.   Return $w$
```

In the next step, the score S of any unmarked instance (i.e., X_t) is calculated using the following definition:

$$S(X_t) = \max_{1\leqslant j\leqslant n}\{w_j\ cdot\ \mu(X_t, X_j), j \neq k\}/\mu(X_t, X_k) \tag{5}$$

The interesting property of the score of an instance X_t (i.e., $S(X_t)$) is that for any value of $w_k > S(X_t)$, X_t will have X_k as its nearest neighbor and thus classified as *ClassT* in LV1 test. This is because the following relation can be easily derived from (5).

$$w_k \cdot \mu(X_t, X_k) > \max_{1\leqslant j\leqslant n}\{w_j\ cdot\ \mu(X_t, X_j), j \neq k\} \tag{6}$$

To find the best value of w_k, unmarked instances are ranked in ascending order of their scores. Assuming that L instances $\{X_1, X_2, \ldots, X_L\}$ are in the list and $S(X_1) < S(X_2) < \ldots S(X_L)$, $L + 1$ values of w_k are examined to find its best value. The first and last values examined are 0 and $S(X_L) + \tau$, respectively (τ is a very small positive number). The rest (i.e., $L - 1$ values) are chosen in the middle of two successive scores in the list. The LV1 classification of instances in the list can be easily calculated for any chosen value of w_k as we know the true and predicted class of each instance. The best value of w_k is the one that maximizes the LV1 classification rate for the instances in the list. Using the specified value of w_k, the algorithm updates the associate lists in the final loop of the algorithm (at line 34). The update is only required for those instances that their score is less than the specified value of w_k. This is done by removing the instance from its current nearest neighbor‘s list of associates and then adding it to the list of associates of X_k.

In order to reduce the time complexity, the WDNN algorithm keeps the associate list, the nearest neighbor (and their distances/similarities) of each training instance in memory.The associate list and the nearest neighbor of each instance are updated as the weights of instances change during the execution of the WDNN algorithm For an instance having m associates, the WDNN algorithm needs $O(mn)$ time to update the associate lists at line 12, where n is the number of training instances. An $O(k\log(k))$ time is required to sort the instances at line 19, where k ($k < n$) is the number of unmarked instances. An $O(k)$ time is required to find the best weight for the instance (i.e., time to test k different values of weight). Finally, an $O(n)$ time is required to update the associate lists at the final loop of the algorithm (at line 34). The overall complexity of the WDNN algorithm for updating the weight of an instance is thus $O(mn + k\log(k) + k + n)$.

Our WDNN algorithm can be viewed as the extended version of the DROP2 algorithm [18], which is a popular instance reduction technique proposed in the literature. In DROP2, the weight assigned to each instance is either 0 or 1. An instance is removed from the training set (i.e., by setting its weight to zero) if it does not hurt the LV1 classification rate on the given training set. On the other hand, the WDNN algorithm attempts to specify the weight of the instance under investigation in such a way that the LV1 classification rate is improved by the maximum amount. An instance is removed from the training set only if the WDNN algorithm is unable to find a weight for that instance that improves the LV1 classification rate.

3.1. Modification of the algorithm for prototype reduction

The WDNN algorithm discussed in the previous section assigns a non-zero weight to an instance if it can improve the LV1 classification of at least one more training instance. In the modified version of the WDNN algorithm (i.e., MWDNN), a non-zero weight is assigned to an instance if it can improve the LV1 classification of at least G training instances. This is achieved by replacing the instruction at lines 25 and 30 of the WDNN algorithm with the following instruction

$$\textit{If current} \geqslant \textit{best} + G \textit{ then}$$

Obviously, the parameter G should be a function of the number of instances in the training set rather than using a fixed value for all datasets. We used $G = 0.03 \times (\textit{no. of training data/no. of classes})$ in the experiments reported in the next section. In fact, the WDNN algorithm is a special case of the MWDNN algorithm that uses $G = 1$.

The parameter G controls the number of final prototypes (i.e., having non-zero weight). When dealing with noisy datasets, it is expected that the MWDNN achieve better compression rates by preventing noisy instances from being included in the reduced prototype set.

It must be noticed that the MWDNN algorithm is no longer a greedy algorithm since the algorithm sets the weight of an instance to zero if it can not improve the LV1 classification of at least G training instances. Therefore, the LV1 accuracy may fluctuate during the execution of the algorithm.

We use the following measure to compare Training Set Compression Rate (TSCR) of various schemes.

$$TSCR = 1 - (r/n) \tag{7}$$

where, n and r represent the original and reduced number of examples, respectively. The basic NN uses all of the training instances (i.e., $r = n$) and has a TSCR of zero.

4. Experimental results

The capabilities of the WDNN and MWDNN algorithms have been empirically assessed through two different types of experiments. In the first one, the proposed algorithm was applied to a two-dimensional problem to show the decision boundaries created with the weighted instances. In the second, a number of tasks from the UCI/Statlog repositories were considered. The aim of these experiments was to show that the proposed technique is uniformly adequate for a variety of tasks involving different data conditions: large/small training sets, continuous/discrete features, large/small dimensionalities, etc.

4.1. Artificial data

An artificial dataset was created in a two-dimensional real space bounded to the region of $[0,1]$ in each dimension. It contains only two output classes in order to make the graphing and visualization easier. Fig. 1 shows the distribution of the data points for this problem. Application of the MWDNN algorithm to this problem reduced this dataset to two instances (i.e., two instances having non-zero weight). These two instances are marked with rectangular boxes in Fig. 1. Using these weighted instances, the resulting boundary between the two classes is also shown in this figure. It must be noted that using these weighted prototypes only five instances are misclassified in LV1 test, while 11 instances are misclassified with the basic nearest neighbor.

This example shows that a much more complex decision surface can be created with fewer instances if instance weights are employed. Without instance weighting, many instances of the outer class (i.e., the class shown with "+") must be stored in order to contain the influence of the stored instances from the inner class.

4.2. Real world problems

We used 14 datasets chosen from UCI-Statlog repository to evaluate the performance of the WDNN and MWDNN algorithms. Table 2 gives a short summary of the datasets used in the experiments.

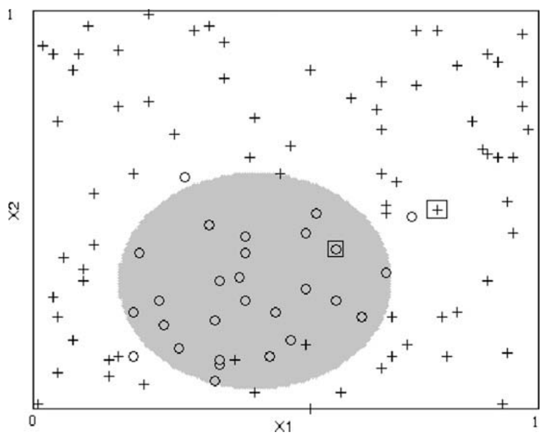

Fig. 1. The result of applying the MWDNN algorithm on an artificial dataset showing the reduced dataset (i.e., the two instances marked with the rectangular boxes) and the resulting decision boundary between the two classes.

Table 2
Statistics of the data sets used in the experiments.

Data set	# of instances	# of features	# of classes	# of non-numeric features
Australian (A)	690	42	2	8
Balance (B)	625	4	3	0
Cancer (C)	685	9	2	0
Diabetes (D1)	768	8	2	0
DNA (D2)	3186	180	3	0
German (G1)	1000	24	2	13
Glass (G2)	214	9	6	0
Heart (H)	270	13	2	7
Letter (L1)	20,000	26	16	0
Liver (L2)	347	6	2	0
Satimage (S)	6435	36	6	0
Vehicle (V1)	846	18	4	0
Vote (V2)	435	16	2	16
Wine (W)	178	13	3	0

For datasets having categorical attributes, each categorical attribute was replaced by P binary attributes. Where, P is the number of different values that the attribute can assume. Five-fold cross validation (5-CV) was used to measure the performance of various schemes. In 5-CV, the dataset is divided into five disjoint groups of approximately equal size. Using four partitions (i.e., 80% of data) as training data, the proposed WDNN and MWDNN methods were used to specify the weights of training instances (using three passes over the data). Using the weighted training instances, classification accuracy on the test partition was measured. This process was repeated until all partitions were uses in the test phase. The 5-CV test was repeated 100 times and average classification rate on test data was calculated. For each data set, we either used the raw Euclidean or Class Dependent Mahalanobis (CDM) metric. The LV1 error-rate on the training data for each of these metrics was used to choose the metric that outperforms the other on that dataset. With this, our experimental setting is the same as the reported results in LPD [11] and PW [12], which enable us to compare our results with results reported in these papers.

In Fig. 2, we have plotted the LV1 error-rate of the classifier during the learning process. The diabetes dataset was used for plotting this figure, which shows LV1 error-rate as the weight of each instance (in one fold of a 5-CV test) is calculated using the WDNN algorithm. Starting with an initial LV1 error-rate of 27.97% (i.e., all instances having equal weight), the WDNN algorithm reduces the error-rate to 10.57% by one pass over the data. It can also be seen that LV1 error-rate never increases during the optimization process.

Fig. 3 shows the above plot when the MWDNN algorithm is used. As seen, the MWDNN algorithm reduces the error-rate from 27.97% to 19.84% by one pass over the data. Fig. 3 also shows that MWDNN is not greedy and the error-rate fluctuates during the learning process.

Fig. 4 shows the previous plots when our proposed algorithms are used for 10 iterations. As seen, the second and subsequent iterations can further reduce the LV1 error-rate. In addition, it can be seen that MWDNN is not as effective as WDNN in reducing the error-rate. However, MWDNN is designed to achieve better compression rates especially when dealing with noisy datasets.

Table 3 gives the average 5-CV generalization accuracy and compression rates of our proposed methods for the datasets used in this paper. For comparison, the classification rate of the basic NN method is also reported in this table. Next to each

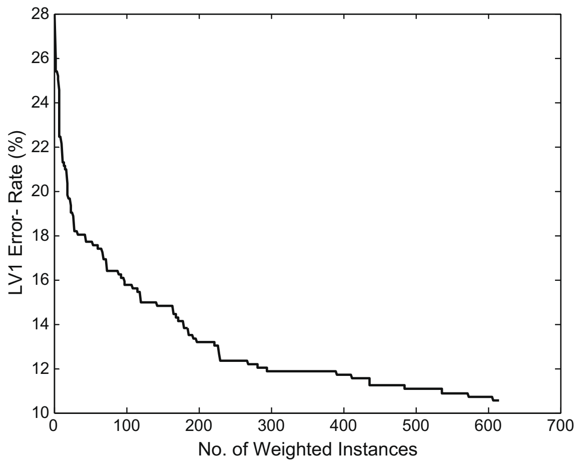

Fig. 2. Leave one out classification error-rate during the first iteration of the WDNN algorithm on diabetes dataset.

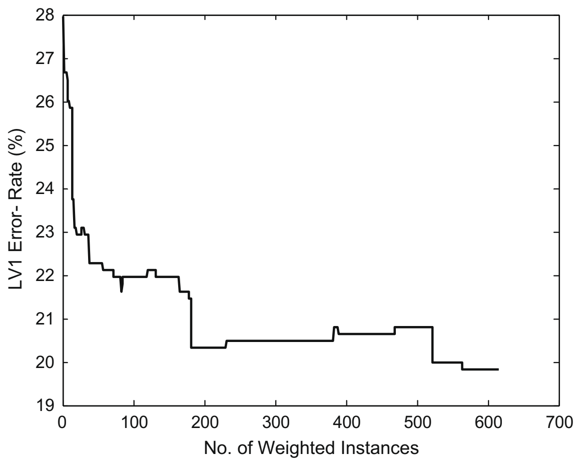

Fig. 3. Leave one out classification error-rate during the first iteration of the MWDNN algorithm on diabetes dataset.

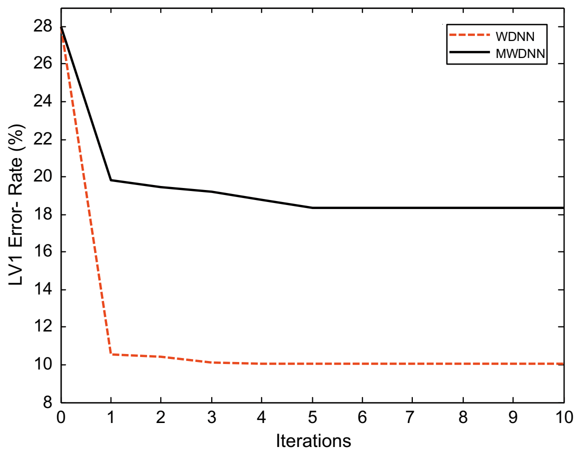

Fig. 4. Leave one out classification error-rate during 10 iterations of the WDNN and MWDNN algorithms on diabetes dataset.

Table 3
The classification and compression rates of our proposed methods on various data sets.

Data Sets	Basic NN	TSCR	WDNN	TSCR	MWDNN	TSCR
Australian (A)	81.36 ± 0.64	0.0	85.48 ± 0.52 (+)	86.37	85.01 ±0.71 (+)	97.34
Balance (B)	69.29 ± 0.73	0.0	89.14 ± 0.81 (+)	91.77	90.32 ±1.18 (+)	98.56
Cancer (C)	96.76 ± 0.45	0.0	97.52 ± 0.31 (+)	97.63	97.88 ±0.66 (+)	98.52
Diabetes (D1)	70.83 ± 0.52	0.0	75.96 ± 0.74 (+)	88.13	76.31 ±1.18 (+)	98.55
DNA (D2)	87.06 ± .074	0.0	96.15 ± 0.62 (+)	84.10	95.84 ±0.68 (+)	96.09
German (G1)	71.13 ± 0.39	0.0	75.84 ± 0.59 (+)	81.04	73.89 ±1.06 (+)	98.81
Glass (G2)	68.26 ± 0.46	0.0	71.34 ± 0.90 (+)	64.56	70.81 ±1.11 (+)	89.29
Heart (H)	76.34 ± 0.34	0.0	83.91 ± 0.26 (+)	84.43	84.91 ±0.33 (+)	96.65
Letter (L1)	94.35 ± 0.51	0.0	97.09 ± 0.63 (+)	89.18	96.81 ±0.58 (+)	97.27
Liver (L2)	64.78 ± 0.58	0.0	68.31 ± 1.15 (+)	71.17	65.39 ±1.29 (+)	96.92
Satimage (S)	88.29 ± 0.36	0.0	89.88 ± 0.45 (+)	91.84	89.01 ±0.51	99.31
Vehicle (V1)	70.43 ± 0.51	0.0	70.14 ± 0.83	70.38	69.15 ±0.97 (−)	97.65
Vote (V2)	92.86 ± 0.53	0.0	92.29 ± 1.09	92.65	91.37 ±1.65 (−)	98.15
Wine (W)	97.29 ± 0.82	0.0	96.61 ± 1.15	92.73	96.04 ±1.54	97.76
Average	80.65 ± 0.69	0.0	84.98 ± 0.91 (+)	84.71	84.48 ± 1.22 (+)	97.21

average accuracy value reported in Table 3, the standard deviation (calculated based on 100 runs of 5-CV) is also given. Using paired T-test (for significance level of 95%), we also report on statistical significance of each case compared to the basic 1-NN method.

Prediction accuracy values that are significantly better or worse than basic NN are marked by (+) and (−) signs, respectively. As seen, the proposed WDNN method improves the generalization accuracy of the basic NN method in 11 out of 14 data sets. Improvements were found to be statistically significant for all of these cases.. The MWDNN method could also improve the generalization accuracy of the basic NN in 11 out of 14 datasets. Improvements were found to be statistically significant for 10 of these cases. In case of vehicle and vote datasets, the performance of the MWDNN method is significantly worse than 1-NN. In the last row of this table, average generalization accuracy of each method over all datasets is reported.

Table 4 gives the average 5-CV generalization accuracy of our proposed methods in comparison with other methods in the literature that attempt to locally adapt the distance measure for the basic NN. The best accuracy result for each dataset is shown in bold. The average accuracy of each method on the entire datasets is also reported in this table. The WDNN method achieves the best average accuracy on the entire datasets.

In order to verify whether differences in accuracies on the entire set of classification tasks are statistically significant, a one tailed Wilcoxon Signed Ranks test [2,3] was used to compare the WDNN and MWDNN algorithms with each of other methods. The confidence level of a significance difference is shown in the last two rows of Table 4 for each of these methods. Positive values indicate the confidence that the WDNN/MWDNN method is "better" than the other methods on these datasets, while negative values indicate the confidence that the WDNN/WDNN method is "worse". Confidence values with a magnitude of at least 90% can be considered significant differences.

As seen in the result of Table 4, both of our proposed methods are significantly better that the A-NN method. This proves the effectiveness of our weight-learning algorithms compared with the heuristic method of [15]. In comparison with LPD [11] and PW [12], both of our methods achieve higher average accuracy on the entire set of classification tasks. However,

Table 4
Classification rates of our proposed methods in comparison with other methods.

Data sets	WDNN	MWDNN	Basic NN	A-NN [15]	PW [12]	LPD [11]
Australian (A)	85.48	85.01	81.36	75.91	83.50	**86.10**
Balance (B)	89.14	**90.32**	69.29	89.88	86.56	83.70
Cancer (C)	97.52	**97.88**	96.76	97.14	96.68	96.60
Diabetes (D1)	75.96	**76.31**	70.83	71.86	72.61	74.00
DNA (D2)	**96.15**	95.84	87.06	92.73	93.51	95.10
German (G1)	**75.84**	73.89	71.13	61.89	71.68	74.00
Glass (G2)	71.34	70.81	68.26	71.22	**73.72**	72.00
Heart (H)	83.91	**84.91**	76.34	67.45	81.06	81.40
Letter (L1)	**97.09**	96.81	94.35	95.87	95.40	96.50
Liver (L2)	**68.31**	65.39	64.78	65.12	63.78	66.70
Satimage (S)	89.88	89.01	88.29	90.49	**91.20**	89.40
Vehicle (V1)	70.14	69.15	70.43	66.28	70.69	**72.60**
Vote (V2)	92.29	91.37	92.86	93.31	94.49	**96.30**
Wine (W)	96.61	96.04	97.29	84.82	**98.65**	95.00
Average	**84.98**	84.48	80.65	80.28	83.82	84.24
Wilcoxon (WDNN)	n/a	+84.2	+99.6	+98.90	+91.6	+82.3
Wilcoxon (MWDNN)	−84.2	n/a	+98.1	+98.10	+56.7	+13.5

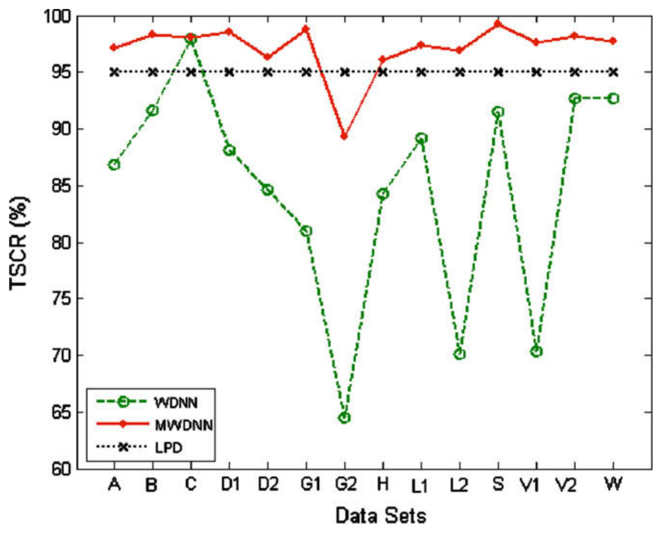

Fig. 5. Compression rates of our proposed methods in comparison with the LPD [11] method.

Table 5
Average accuracies and compression rates of our proposed methods in comparison with the basic NN method in presence of noisy data.

Algorithm	Clean	TSCR	Noisy	TSCR (%)
Basic NN	80.65	0.0	74.45	0.0
WDNN	84.98	84.71	80.42	81.64
MWDNN	84.48	97.21	**81.00**	**96.52**

the differences in accuracies are not statistically significant (except for the WDNN method that is significantly better than the PW method).

In Fig. 5, the compression rates of our proposed method and the LPD [11] method are shown for various datasets used in this paper. It must be noted that the A-NN [15] and PW [12] methods do not reduce the size of the training set.

In the LPD method, the user should specify the required number of prototypes as input to the algorithm. The results of Fig. 5 correspond to the case that 5% of data is selected (as prototypes) by the LPD algorithm. As seen, The MWDNN algorithm achieves the best compression rates on all datasets except for the glass dataset. Overall, the results of Table 5 and Fig. 5 confirm that the MWDNN algorithm can achieve better compression rates (and comparable/better accuracy) when compared with the LPD method.

4.3. *Effect of noise*

The aim of this experiment is to evaluate the performance of the proposed methods in the presence of noise. For this purpose, the same experiments were repeated with 20% *uniform class noise* artificially added to each dataset. This was done by randomly changing the output class of 20% of the instances in the training set to an incorrect value (with equal probability for each of the incorrect classes). The output class of the instances in the test set are not noisy. So the result indicate how well each model is able to predict the correct output even some of its training data is mislabeled.

Table 5 shows the average accuracy and compression rates of each method (including the basic NN method) over all 14 datasets used in this paper.

In presence of noise, the average accuracy of the basic NN dropped 6.2% while the average accuracy of the WDNN and MWDNN methods dropped 4.56% and 3.48%, respectively. Both of these methods achieved accuracy higher than the basic NN method. The WDNN algorithm had the highest accuracy among different methods.

In the presence of noise, the compression rate of the WDDN method dropped 3.07% while the drop in compression rate was only 0.66% for the MWDNN method. The WDNN algorithm achieved the best compression rate among different methods.

Overall, in presence of noisy data, the WDNN algorithm achieves the best compression rate and accuracy among different methods.

5. Conclusions

In this paper, we introduced a new technique for adapting the distance function in NN classification method. This was achieved by assigning a weight to each training instance. We proposed a greedy algorithm for learning the weights of training instances. The learning algorithm attempts to maximize LV1 classification rate, which is the true estimate of the generalization ability of the classifier. Simulation results proved that the scheme could improve the generalization ability of basic NN rule.

We also proposed the MWDNN method as a powerful prototype selection algorithm. In comparison with other prototype selection algorithms proposed in the literature, we showed that the MWDNN method achieves better compression rates and comparable/better accuracy results. We also showed that the MWDNN performs quite well in the presence of noisy data.

References

[1] G. Chen, C.Z. Wang, Q.H. Hu, A new approach to attribute reduction of consistent and inconsistent covering decision systems with covering rough sets, Information Sciences 177 (2007) 3500–3518.
[2] W.J. Conover, Practical Nonparametric Statistics, John Wiley, 1971. pp. 206–209, 383.
[3] M.H. DeGroot, Probabolity and Statistics, second ed., Addision Wesley, 1986.
[4] C. Domeniconi, J. Peng, D. Gunopulos, Locally adaptive metric nearest neighbor classification, IEEE Transaction on Pattern Analysis and Machine Intelligence 24 (2002) 1281–1285.
[5] C.F. Eick, A. Rouhana, A. Bagherjeiran, R. Vilalta, Using clustering to learn distance functions for supervised similarity assessment, Engineering Applications of Artificial Intelligence 19 (4) (2006) 395–401.
[6] J. Friedman, Flexible Metric Nearest-neighbor Classification, Technical Report 113, Department of Statistices, Stanford University, 1994.
[7] T. Hastie, R. Tibshirani, Discriminant adaptive nearest neighbor classification, IEEE Transaction on Pattern Analysis and Machine Intelligence 18 (6) (1996) 607–615.
[8] Q. Hu, D. Yu, J. Liu, C. Wu, Neighborhood rough set based heterogeneous feature subset selection, Information Sciences 178 (2008) 3577–3594.
[9] N. Jankowski, M. Grochowski, Comparison of instances selection algorithm I: algorithms survey, in: L. Rutkowski et al. (Eds.), ICAISC, Springer-Verlag, LNAI 3070, 2004, pp. 598–603.
[10] B.D. Morring, T.R. Martinez, Weighted instance typicality search (WITS): a nearest neighbor data reduction algorithm, Intelligent Data Analysis 7 (6) (2003).
[11] R. Paredes, E. Vidal, Learning prototypes and distances: a prototype reduction technique based on nearest neighbor error minimization, Pattern Recognition 39 (2006) 180–188.
[12] R. Paredes, E. Vidal, Learning weighted metrics to minimize nearest-neighbor classification error, IEEE Transaction on Pattern Analysis and Machine Intelligence 28 (7) (2006) 1100–1110.
[13] A. Rozsypal, M. Kubat, Selecting representative examples and attributes by a genetic algorithm, Intelligent Data Analysis 7 (4) (2003).
[14] Ö. Uncu, I.B. Türkşen, A novel feature selection approach: combining feature wrappers and filters, Information Sciences 177 (2007) 449–466.
[15] J. Wang, P. Neskovic, L.N. Cooper, Improving nearest neighbor rule with a simple adaptive distance measure, Pattern Recognition Letters 28 (2007) 207–213.
[16] J. Wang, P. Neskovic, L.N. Cooper, Neighborhood selection in the k-nearest neighbor rule using statistical confidence, Pattern Recognition 39 (2006) 417–423.
[17] D.R. Wilson, T.R. Martinez, An integrated instance-based learning algorithm, Computer Intelligence 16 (1) (2000) 1–28.
[18] D.R. Wilson, T.R. Martinez, Reduction techniques for exemplar-based learning algorithms, Machine Learning 38 (3) (2000) 257–286.
[19] N. Zeidat, C.F. Eick, Z. Zhao, Supervised Clustering: Algorithms and Applications, Technical Report UH-CS-06-10, Department of Computer Science, University of Houston, USA, 2006.
[20] M.J. Zolghadri, E.G. Mansoori, Weighting fuzzy classification rules using receiver operating characteristics (ROC) analysis, Information Sciences 177 (11) (2007) 2296–2307.
[21] M. Zolghadri Jahromi, M. Taheri, A proposed method for learning rule weights in fuzzy rule based classification systems, Fuzzy Sets and Systems 159 (4) (2008) 449–459.