On the evolutionary optimization of *k*-NN by label-dependent feature weighting

Daniel Mateos-García, Jorge García-Gutiérrez, José C. Riquelme-Santos

A B S T R A C T

Different approaches of feature weighting and *k*-value selection to improve the nearest neighbour technique can be found in the literature. In this work, we show an evolutionary approach called *k*-Label Dependent Evolutionary Distance Weighting (kLDEDW) which calculates a set of local weights depending on each class besides an optimal *k* value. Thus, we attempt to carry out two improvements simultaneously: we locally transform the feature space to improve the accuracy of the *k*-nearest-neighbour rule whilst we search for the best value for *k* from the training data. Rigorous statistical tests demonstrate that our approach improves the general *k*-nearest-neighbour rule and several approaches based on local weighting.

Keywords:
Feature weighting
Evolutionary computation
Label dependency

The use of weight-based models is common in machine learning, and specifically, in classification problems (Wettschereck et al., 1997). The proper adjustment of weights during the training phase improves the model prediction rate. Weighted neural networks are perhaps the most used example (Li et al., 2012; Park, 2009), but support vector machines (Shen et al., 2012) and nearest-neighbour (Chen et al., 2009) methods can also use weights for a better fit. All the weighting proposals have the search for optimal values (in training) in common, attempting to avoid overfitting. This optimization can be carried out by analytical methods (e.g., RELIEF method (Sun, 2007)) or by heuristics such as evolutionary computation (AlSukker et al., 2010) or tabu search (Tahir et al., 2007).

A survey of the research in the literature on weight-based models shows that most of them use a set of weights for all instances. For example, in the weighted nearest neighbour method, a ω_i value is wanted for each feature f_i and represents the influence of f_i in the calculation of every distance which is modified according to the influences of every feature. A weight ω_i is therefore a measure of the importance of f_i.

Although most approaches search for a global set of weights for all instances, there are also other local proposals which use weights mainly applied to prototypes (region) (Fernandez and Isasi, 2008) or class labels (AlSukker et al., 2010; Chen et al., 2011). In this work, the latter option has been analyzed, taking into account that optimizing a model with a set of weights for each prototype may prove to be too hard (in terms of execution time) for evolutionary computation.

We explore the hypothesis that feature influence is not the same for every label. For example, if we classify patients according to three labels representing variants of a disease, the feature age could be more important for one variant than for another, and therefore, it could be more appropriate to have different weights depending on the variant. The difficulty of such a model is obvious: if we want to classify a new example with an unknown class and we have different weights for each class, which set of weights should we apply? Hence, we propose a heuristic based on an evolutionary algorithm that firstly searches for a set of different weights per class or label in the training phase and later selects the best set of weights for each instance in the testing phase to optimize a nearest neighbour classifier. Moreover, the selection of the number of neighbours (*k* parameter) for the nearest neighbour classifier is essential for its best performance. For this reason, we also use the evolutionary search to simultaneously attempt to find the best *k*.

In the next sections, we show how the combination of weight matrices and a proper number of neighbours results in an objective good performance of the nearest neighbour rule. The rest of the paper is organized as follows. Section 2 presents related works. Section 3 describes the general process for obtaining and using the weight matrix in the nearest neighbour method. The results achieved are shown in Section 4. Finally, Section 5 presents a summary of the conclusions and future lines of work.

2. Related work

There are many works in the literature which refer to how to obtain weights to improve machine learning techniques. These

works can be classified depending on the learning model and the optimization algorithm used to find the weights. Since our work is based on the nearest neighbour (NN) rule, we only present proposals based on this paradigm in this section. As we have mentioned above, weighting can refer to a global or a local scope. With regard to global methods, Raymer et al. (2000) present a proposal to select and remove features by a genetic algorithm in combination with NN. This algorithm optimizes a vector of weights that is used to scale the original features. They also use another bit vector to make a feature selection simultaneously. In a later work, Raymer et al. (2003) show a hybrid evolutionary algorithm based on the Bayes discriminant function for isolating characteristics belonging to large datasets of biomedical origin by selecting and extracting features. In addition to evolutionary-based optimization methods, we can find other heuristics. Thus, Tahir et al. (2007) present a hybrid approach to simultaneously select and weight features through tabu search (TS) and improve the classification accuracy of *k*-NN.

Considering prototype weighting, Fernandez and Isasi (2008) propose a local system used with a prototype-based classifier. The weights are iteratively calculated after applying a local data normalization. This normalization is based on the positions of the instances with respect to the prototype (or region) which they belong to.

Following with local methods, AlSukker et al. (2010) use a Differential Evolution (DE) optimization technique to find weights that affect different elements of the training set. Four proposals are described in their work: feature weighting, neighbours weighting, classes weighting and mixed weighting (features and classes), and the best results are obtained for the latter. Thus, the mixed method obtains a vector of weights by concatenating two: a vector of weights by feature, and a vector of weights by class. Therefore, the degrees of freedom of this proposal are given by the dimension of the resultant concatenated vector (number of features + number of classes). Mohemmed and Zhang (2008) present a Nearest Centroid Classifier (NCC) based on the search of the arithmetic mean of classes from the training data. The instances with unknown label are classified by measuring the distance to the calculated means. Particle swarm optimization is used to find the centroids that minimize the classification error, and one of the similarity functions used in this paper consists of a label-dependent Euclidean distance. This function is introduced by Paredes and Vidal (2006). The similarity function (which is used in our work) is a weighted distance with the objective of improving the performance of NN. A first approach considers a weight for each feature and instance from the training data, resulting in a non-viable number of parameters in the learning process. Therefore, the authors present three types of reduction: a weight by class and feature (class-dependency), a weight by prototype (prototype-dependency) and a combination of the previous two. The optimization step is carried out by Gradient Descent.

There are also references to unbalanced data processing. Liu and Chawla (2011) define a Class Confidence Weight (CCW) as the probability of a feature value given a class. The CCW estimation is calculated by mixture models for numeric features, and Bayesian networks for categorical attributes.

As we have seen previously, a set of weights can be applied in a global or local way for the nearest neighbour rule. In this work, the latter option has been analyzed to achieve two goals: getting a trade-off between complexity and best fit, and objectively showing the effectiveness of a local feature weighting method. For the first goal, we have chosen an evolutionary approach to obtain a matrix of weights (a weight for each class and feature, and therefore with as many degrees of freedom as the matrix dimension, i.e. number of features $\times$ number of classes) and the best value of the *k* parameter for the *k-nearest neighbour*. For the second goal, we have evaluated the experimental results with rigorous statistical tests that have already been used in the literature (Demšar, 2006; García and Herrera, 2008). All these aspects are exposed in the next sections.

3. Method

In this section, we describe our method of feature weighting named *k-Label Dependent Evolutionary Distance Weighting* (kLDEDW). Section 3.1 presents the aim of this approach and the use of the learned parameters (weights and number of neighbours). Section 3.2 details the search algorithm itself.

3.1. Purpose and functionality

As noted above, we aim to find a set of weights and a proper *k* to optimize the nearest neighbour rule. The basis of our method is that this set of global weights will not be the same for all instances but will depend on the label or class instance. Therefore, we are searching for a matrix of weights instead of the usual vector. This matrix will have as many rows as labels in the training file, and as many columns as features. In this way, we obtain weights for each feature and label, which allows us to transform the feature space for a later classification. In addition, this transformation will be driven by the best *k* parameter found for the considered training data.

A set of weights by label provides more information than a single weight vector. However, it raises the problem of selecting which set of weights (matrix row) has to be applied in the testing phase when we do not know the class of any given instance. Therefore, a heuristic is needed to determine the weights used to classify a new testing example. This heuristic is based on the nearest neighbour rule but with a modified distance calculation. To simplify the notation, we assume that the set of classes or labels can be represented by the set of integers between 1 and the number of labels *b*. Therefore, let $D = \{(e,l)|e \in \mathbb{R}^f \text{ and } l \in \{1,2,\ldots,b\}\}$ be our dataset, with f features and b labels from different classes. Let *label* be a map that assigns to each element e its class. Suppose that we divide D into two sets TR and TS, corresponding to training and testing sets, respectively, so that $D = TR \cup TS$ and $TR \cap TS = \emptyset$. In this way, examples of TS (testing set) will be used to test the goodness of kLDEDW, and they will not play a role in obtaining the weights (see Fig. 1 where testing data belongs to a different set than training data). As we will detail in Section 3.2, we obtain a matrix $W = (\omega_{ij})_{b\times f}$ from only the examples of TR. This matrix will be used to modify the calculation of the distance between an instance $x \in TR$ and another one $y \in TS$ during the testing phase.

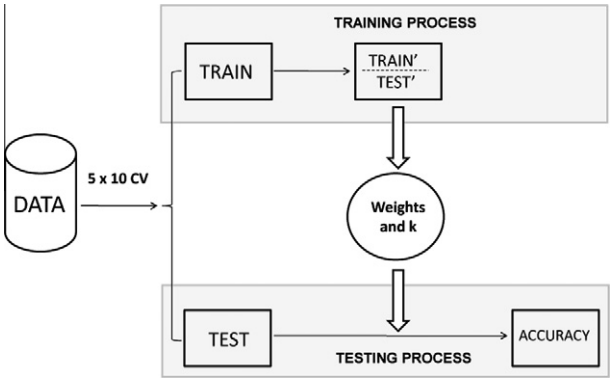

Fig. 1. Training and testing processes.

For this purpose, we use a weighted distance d_w (Paredes and Vidal, 2006) defined as follows:

$$d_w(x,y) = \sum_{k=1}^{f} \omega_{label(x)k}(x_k - y_k)^2 \quad \text{with } x \in TR,\ y \in TS$$

$\omega_{label(x)k}$ is the element (weight) of the weighting matrix corresponding to the row $label(x)$ and the column k (1)

The justification for this distance is given by the dependence of the weights on the class so that the calculation of the distance of a testing example y to the training example x depends on the label of x. As we can see in Eq. (1), the row selection of the weight matrix is determined by the label of the training example. Thus, once this distance d_w is defined, the classification of an example of TS is carried out by following the nearest neighbour rule but calculating the distance with d_w.

Consider an example. Suppose that we have six training examples belonging to three different classes and two features. Let $x_1 = (1.2, 2.3, 1)$, $x_2 = (3.2, 1.3, 1)$, $x_3 = (4.3, 3.3, 2)$, $x_4 = (3.2, 4.7, 2)$, $x_5 = (3.2, 4.3, 3)$, $x_6 = (5.2, 1.3, 3)$ be the examples. The first two values represent the features, and the third value is the class (1, 2 or 3). Let the matrix W be $((1.4, 2.1), (3.1, 0.2), (0.7, 0.4))$; if we want to find the label of the point $y = (3.2, 4.6)$, we would calculate the distance to y from the six training examples, so that:

$d_w(x_1, y) = 1.4(1.2 - 3.2)^2 + 2.1(2.3 - 4.6)^2 = 16.709$
$d_w(x_2, y) = 1.4(3.2 - 3.2)^2 + 2.1(1.3 - 4.6)^2 = 22.869$
$d_w(x_3, y) = 3.1(4.3 - 3.2)^2 + 0.2(3.3 - 4.6)^2 = 4.089$
$d_w(x_4, y) = 3.1(3.2 - 3.2)^2 + 0.2(4.7 - 4.6)^2 = 0.002$
$d_w(x_5, y) = 0.7(3.2 - 3.2)^2 + 0.4(4.3 - 4.6)^2 = 0.036$
$d_w(x_6, y) = 0.7(5.2 - 3.2)^2 + 0.4(1.3 - 4.6)^2 = 7.156$

As the minimum value is 0.002, for $k = 1$ we determine that the class of y is the same as x_4 (class 2). If we consider $k = 3$, the nearest points are x_4, x_5 and x_3, and the majority class is then chosen (also class 2). This classification rule is formalized in the function *NearestN* detailed in Section 3.2.3.

3.2. Calculation of weights and k

In this subsection, we detail the evolutionary search algorithm for obtaining the weighting matrix and the optimal k value. To determine an evolutionary algorithm, it is necessary to define an individual encoding, genetic operators, a fitness function and a rule to carry out the generational replacement. These aspects are described in the following subsections.

3.2.1. Individual encoding

An individual is a set of weights for each label and a value for k; the weights are represented by a matrix with one row for each label and a column for each feature. Each element of the matrix is a real value whose initial lower and upper bounds can be customized. In our case, the chosen range is $[0, 1]$ so that, initially, a value of 0 indicates that the corresponding feature is irrelevant and a value of 1 indicates that it is indispensable. However, it is possible to reach values above one through crossover and mutation operators. For the number of neighbours the chosen range is an odd integer in $[1, 5]$. Therefore, the initial population consists of N real-valued matrices of dimensions $b \times f$ with random values between 0 and 1, and N random odd integers from 1 to 5.

3.2.2. Genetic operators

Since our algorithm works with matrices instead of the usual linear structures, we have adapted the operators of crossover and mutation used in our individuals. Thus, the crossover between two matrices is carried out row by row so that the ith row of one parent crosses the ith row of the other. For this crossing between two vectors of real numbers, we use the *BLX-α* crossover that is very common in the literature (Eshelman and Schaffer, 1993). For the k parameter, the operation consists of the random selection between each one of the k values from the ancestors. The *BLX-α* crossover is described as follows. If ω_{ij} and ω'_{ij} are the jth elements of the ith row of each parent, the new gene is a real value randomly chosen in the range $[\omega_{min} - I\alpha, \omega_{max} + I\alpha]$ (Fig. 2).

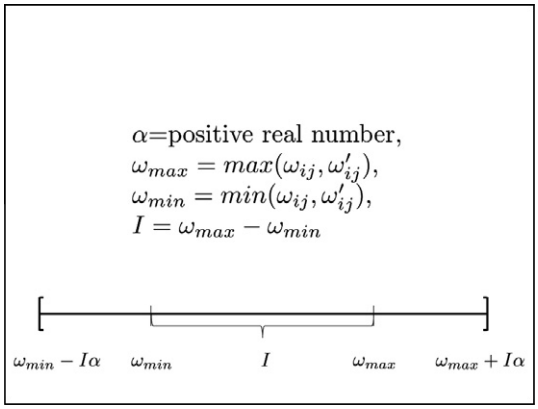

Fig. 2. *BLX-α* crossover.

Regarding the weighting matrix and the mutation operator, each row has a probability of p that any of its weights w is increased or decreased in a random value $\delta * w$. Initially, $\delta \in [0, 1]$, but for a better fit, every g generations, the upper bound is reduced by g/G, where G is the total number of generations. For example, if $G = 100$ and $g = 10$, during the first ten generations, the upper bound for δ is 1, in the following ten it will be 0.9, in the next ten 0.8, and so on. For the k parameter, its value is increased or decreased in two unities in the range $[1, 5]$ with a probability of p.

3.2.3. Fitness function

During the training phase, we only use the subset $TR \subset D$. The fitness function should reward those individuals that obtain a better classification on the training set but that prevent over-fitting for these examples. This is the reason why the training phase and the fitness function are based on a cross-validation on TR and its accuracy respectively. We have to keep in mind that we can know the class label on the TR data, and thus the fitness works as a measure of the quality of the performed prediction.

As we show in Fig. 3, the fitness is calculated using $m \times s$ cross-validations, where m is the number of times to repeat the validation process (line 3) and s is the number of partitions of the training set TR (line 4). Thus, for each validation, the set TR is divided into s bags $B_1, B_2, \ldots, B_s$. For each bag B_j, we evaluate the fit of a classification, where B_j acts as the testing set and the remainder of TR as the training set. This evaluation is conducted by the *Evaluate* function, which will be analyzed later. The error of this classification on B_j is accumulated on average (lines 7 and 9) by the *partialError* variable for each bag, and by *error* (line 10) for each validation. Finally, the fitness value corresponds to the mean value of the error for all validations (line 12).

The *Evaluate* function takes as input parameters the matrix W, the k value, the virtual training set $TR - B_j$ and the virtual testing set B_j (line 7) and returns the accuracy for the virtual testing set of a classifier based on W taking the reference instances from the virtual training set. The classifier used is a version of the k nearest neighbour based on the modified Euclidean distance from Eq. (1). For each virtual testing instance (line 16), the label returned is the majority label corresponding to its k nearest neighbours in the virtual training set by the *NearestN* function (line 17). If the label returned does not match the label of the virtual testing

```
1: Fitness(W, k, TR) : error
2: error = 0
3: for i = 1 to m do
4:    Divide TR into s bags: B_1...B_s
5:    partialError = 0
6:    for j = 1 to s do
7:       partialError = partialError + Evaluate(W, k, TR − B_j, B_j)
8:    end for
9:    partialError = partialError/s
10:   error = error + partialError
11: end for
12: error = error/m
13: return error

14: {note that in Evaluate function, Train and Test are subdivisions from original training data
    (the real testing set is unknown at this moment and it is not used in the fitness calculation)}
    Evaluate(W, k, Train, Test) : error
15: error = 0
16: for each instTest in Test do
17:    lab = NearestN(W, k, Train, instTest)
18:    if lab ≠ label(insTest) then
19:       error = error + 1
20:    end if
21: end for
22: error = error/size(Test)
23: return error

24: NearestN(W, k, Train, y) : labY
25: sortedInst and kneighbours are empty Sorted Sets
26: for each x in Train do
27:    insert x into sortedInst ordered by d_w(x, y)
28: end for
29: kneighbours = sortedInst.get(k)
30: labY = majorityLabel(kneighbours) {if there are ties, labY =
    label(sortedInst.getFirst())}
31: return labY
```

Fig. 3. Fitness function.

example, the error increases by 1 (line 19). Subsequently, the resulting error is normalized by the virtual testing set size (line 22). Therefore, the value returned by *Evaluate* is a real value between 0 (classification success) and 1 (misclassification). The *NearestN* function calculates the closest instances from the virtual training set to the virtual testing instance *y* (line 24 et seq.). We insert each example from the virtual training data in a sorted set according to its distance to *y* (line 27) so that the resulting first

example of the sorted set is the nearest neighbour, and the last is the farthest neighbour from y. After selecting the k nearest neighbours from the sorted set (line 29), the majority label is returned (lines 30 and 31).

3.2.4. Generational replacement

For the replacement of individuals from one generation to the next, we have chosen an elitist design where the best individual moves to the next generation without being affected by the mutation operator. The rest of the new population is formed as follows: if the number of individuals is N, the following $C-1$ individuals are formed by cloning the best individuals of the previous generation. The remaining $N-C$ individuals are formed by the crossover operator. All individuals except the first are subjected to a mutation operator with probability p. To select the individuals that will be involved in the crossover operation, the roulette-wheel and tournament methods have been tested without any significant differences.

4. Results

This section presents the algorithm setup and the results obtained by our proposal compared with other algorithms. In Section 4.1, we analyze the parameters of our approach, in Section 4.2, we show the results, and in 4.3, these results are statistically validated.

4.1. Parameters

The configuration of standard evolutionary algorithms is widely discussed in the literature but for exclusive parameters it is necessary to perform an analysis of its behaviour. To observe the parameters sensibility of our approach, we performed eight experiments for four different datasets. Four experiments were made by varying the parameter α of the crossover operator, and the remaining by varying the parameter g of the mutation operator. As seen in Section 3.2.2, the parameter α is related to the *BLX*-α operator for real-coded individuals, and g is a parameter ad hoc. For each experiment the average of the accuracy was obtained using 2×2 cross-validation a total of three times (with the same folders). We can see that although the influence of the parameters is not significant, for a value of $g=20$ we see a better overall performance (see Table 1). The parameter α seems to be slightly less influential, although the literature provides a standard value of 0.5. Taking into account the above, the default setting for the studied parameters is $\alpha=0.5$ and $g=20$. All other values of our evolutionary algorithm are those that appear widely in the literature, i.e. a population size of 100, 10% of elitism (value of C) and a probability of mutation of 10%. Regarding the number of generations, tests showed stability in the fitness value when 90–100 iterations were achieved, so that 100 was took as default value.

Table 1
Sensibility analysis for α and g. Bold values are the best results for each dataset.

	Balance s.	Haberman	Liver d.	Tae
$\alpha=0.3$	86.704	73.148	60.203	55.665
$\alpha=0.4$	86.704	**73.910**	58.901	56.651
$\alpha=0.5$	86.491	73.529	**60.829**	**56.765**
$\alpha=0.6$	**86.944**	72.386	60.059	56.217
$g=5$	86.678	72.277	60.926	55.995
$g=10$	86.997	72.985	61.409	56.659
$g=15$	87.317	72.494	**61.552**	54.124
$g=20$	**87.343**	**73.802**	61.312	**58.855**
Average	**86.897**	**73.066**	**60.649**	**56.366**
Standard deviation	**±0.310**	**±0.643**	**±0.890**	**±1.32**

To make a fair comparison with other algorithms in the experimentation step, we have considered the same configuration of our approach in the following subsections.

4.2. Experiments

We can see in Table 2 the results obtained by the different approaches referenced in Section 2 with datasets from the UCI repository (Asuncion and Newman, 2007). As we have seen previously, the represented works have in common the use of local weighting. Because there is no concordance in validation methods and data used by each approach, this table is indicative only and it can not be employed as a valid comparative study. We can observe that the most complete experimentation is performed by Paredes and Vidal (2006).

To perform the experiments and a proper comparative study, we have implemented DE4 (AlSukker et al., 2010), and compiled the source code of CW (Paredes and Vidal, 2006) which is available on http://users.dsic.upv.es/∼rparedes/research/CPW/index.html. Also, we have compared kLDEDW with IBk (implementation of k nearest neighbour in the WEKA framework (Hall et al., 2009) for three typical values of k: 1, 3 and 5. Thirty datasets from the UCI repository were used with different numbers of labels and types of features. As will be seen later, this number of tests allows us to make a reliable statistical analysis. We have chosen to use a 10 cross-validation as testing method. To reduce the influence of the random nature of the evolutionary algorithm, for each dataset, five experiments were carried out, each with a different data ordering. Therefore, we had a total of 50 experiments by dataset (5×10 cross-validation). The data used in this experimentation were pre-processed by the WEKA framework (Hall et al., 2009). Thus, nominal features were converted to numerical values, all values were normalized to lie between zero and one, and finally all numerical missing values were replaced by the mean value of the corresponding feature. All algorithms involved in experiments were tested with the same pre-processed datasets (see Table 3).

Table 3 shows the percentage results for the 30 datasets used. Each row contains the mean accuracy of 50 runs for each classifier (5×10 CV). We can observe for the IBk family that a high value of k does not necessarily imply a better accuracy, and therefore the selection of a proper k is important for a good performance of nearest neighbour rule. At the foot of Table 3, we show the mean success of all tests. An improvement can be seen in the results when our proposal is compared with its competitors. On the one hand, we see that our work's averages are better than the other proposals and its standard deviation is close to that obtained by the IBk family but lower than of the two proposals based on local weighting. Furthermore, our method wins 11 out of 30 data sets, and it is second in 13 of the remaining 19. In the next subsection, we show that this improvement is significant from the statistical point of view.

4.3. Statistical analysis

To establish a proper evaluation of the results, we have conducted a series of tests that statistically validate the comparison of all of the algorithms used in the experimentation. Specifically, we have used the non-parametric Friedman's test and Holm's post hoc procedure. We use a non-parametric test because of the failure of some of the conditions for the application of parametric tests (independence, normality and homoscedasticity) (Demšar, 2006; García and Herrera, 2008). Concretely, Demšar (2006) suggests that the condition of independence cannot be verified in cross-validations because a portion of the examples is used both for training and testing in different partitions. In our case, having used a 5×10 cross-validation, the condition of independence is therefore not satisfied.

Table 2
Accuracy of related works.

Validation	kLDEDW 5 × 10 CV	Paredes and Vidal (2006) 100 × 5 CV	Mohemmed and Zhang (2008) 10 × 3 CV	Fernandez and Isasi (2008) 100 × 10 CV	AlSukker et al. (2010) 10 × Hold-Out (66%-33%)
Aaustralian	85.943	82.63	85	–	–
Balance s.	88.186	82.02	90.7	–	–
Breast w.	96.529	96.31	96.2	–	99.93
Diabetes	74.533	72.01	75.8	74.118	81.04
Glass	73.593	71.48	–	63.714	–
Heart s.	79.47	77.66	81.2	–	–
Ionosphere	92.207	–	–	–	98.85
Liver d.	61.004	59.78	–	65.294	–
Sonar	86.466	–	–	–	90.19
Vehicle	72.43	70.62	57.7	–	–
Vote	95.651	93.39	–	–	–
Vowel	98.815	98.64	–	–	–
Wine	97.913	98.56	93.8	–	–

Table 3
Accuracy of each studied algorithm. Bold values are the best results for each dataset.

	kLDEDW	IB1	IB3	IB5	CW	DE4
Australian	**85.94 ± 2.04**	80.20 ± 2.25	83.55 ± 1.92	84.37 ± 1.24	80.92 ± 4.27	81.88 ± 2.20
Balance s.	88.18 ± 1.00	86.80 ± 0.97	86.90 ± 1.10	**88.28 ± 0.95**	85.63 ± 3.22	58.40 ± 6.59
Breast t.	**71.01 ± 5.50**	68.29 ± 5.01	63.90 ± 5.98	65.39 ± 7.65	70.74 ± 10.07	67.54 ± 3.71
Breast w.	96.52 ± 1.16	95.65 ± 1.05	96.63 ± 0.98	**97.18 ± 1.08**	96.75 ± 2.15	96.34 ± 1.31
Car	**96.38 ± 0.59**	93.12 ± 0.59	93.12 ± 0.59	93.12 ± 0.59	73.18 ± 29.87	96.32 ± 0.67
CMC	46.89 ± 2.32	44.35 ± 1.37	**47.05 ± 1.68**	45.93 ± 1.58	44.01 ± 4.04	44.85 ± 1.65
Diabetes	74.53 ± 2.04	70.93 ± 2.10	74.38 ± 2.21	**74.72 ± 1.54**	68.74 ± 3.99	67.07 ± 2.51
E. coli	86.13 ± 2.11	80.23 ± 2.84	84.83 ± 2.19	**86.48 ± 1.96**	80.53 ± 5.93	79.91 ± 2.89
Glass	73.59 ± 3.41	70.01 ± 3.33	68.62 ± 3.41	66.12 ± 4.64	**75.23 ± 7.4**	64.94 ± 6.39
Haberman	**74.40 ± 2.60**	67.03 ± 2.59	71.58 ± 2.71	71.07 ± 1.77	71.51 ± 4.55	56.01 ± 3.96
Heart s.	**79.47 ± 2.78**	75.20 ± 3.11	78.52 ± 3.39	78.36 ± 3.13	76.22 ± 6.94	74.37 ± 3.26
Hill v.	**53.32 ± 2.26**	50.31 ± 1.52	51.16 ± 2.40	51.39 ± 2.58	52.86 ± 5.34	52.27 ± 2.65
Ionosphere	**92.20 ± 2.18**	86.89 ± 2.46	86.13 ± 1.78	85.61 ± 1.52	91.68 ± 4.63	91.97 ± 2.11
Liver d.	61.00 ± 3.78	59.31 ± 3.96	61.80 ± 3.85	58.35 ± 3.70	**63.33 ± 7.96**	62.54 ± 3.43
Lymphoma	**85.12 ± 5.69**	81.73 ± 3.03	78.70 ± 4.14	78.53 ± 4.30	84.85 ± 8.93	79.86 ± 4.11
Mammographic m.	81.04 ± 1.69	76.83 ± 1.89	80.97 ± 1.84	**82.20 ± 1.52**	75.96 ± 4.07	76.07 ± 1.66
Mfeat m.	**72.04 ± 0.91**	65.83 ± 1.49	69.65 ± 1.30	71.08 ± 1.15	65.99 ± 2.57	66.63 ± 1.02
Ozone	94.00 ± 0.26	92.27 ± 0.95	93.96 ± 0.30	**94.04 ± 0.31**	93.80 ± 0.61	78.06 ± 1.24
Pendigits	99.49 ± 0.04	99.36 ± 0.12	99.37 ± 0.08	99.27 ± 0.08	**99.57 ± 0.17**	99.43 ± 0.04
Postoperative p.	69.47 ± 4.28	62.89 ± 3.08	69.24 ± 4.31	**72.40 ± 3.87**	63.77 ± 13.61	43.80 ± 9.79
Sonar	86.46 ± 3.78	84.85 ± 3.44	83.00 ± 4.93	82.56 ± 3.68	**88.97 ± 6.60**	85.02 ± 2.69
Sponge	90.86 ± 3.88	92.33 ± 3.08	88.78 ± 1.74	88.78 ± 1.74	**95.00 ± 6.18**	87.14 ± 7.11
Tae	61.33 ± 6.63	60.96 ± 6.14	50.35 ± 7.71	53.65 ± 5.86	**66.88 ± 12.17**	63.68 ± 7.56
Transfusion	**76.34 ± 1.25**	69.46 ± 2.08	73.81 ± 1.50	75.99 ± 1.65	68.05 ± 15.39	64.24 ± 2.26
Vehicle	72.43 ± 1.72	70.00 ± 1.44	70.59 ± 1.58	70.91 ± 1.53	**72.47 ± 3.92**	71.02 ± 1.73
Vote	**95.65 ± 1.67**	93.00 ± 1.56	93.95 ± 1.84	94.01 ± 2.31	95.31 ± 3.07	94.39 ± 1.32
Vowel	98.81 ± 0.62	99.07 ± 0.34	96.41 ± 1.46	92.75 ± 1.35	99.23 ± 0.87	**99.30 ± 0.30**
Wine	97.91 ± 1.92	94.51 ± 1.84	95.64 ± 2.29	95.48 ± 2.35	**97.99 ± 3.51**	97.48 ± 2.08
Yeast	57.24 ± 1.15	52.91 ± 1.43	55.21 ± 1.13	**57.56 ± 1.38**	53.47 ± 2.83	50.98 ± 1.55
Zoo	91.08 ± 3.41	95.37 ± 2.82	92.73 ± 2.79	94.66 ± 2.17	**96.07 ± 5.93**	94.24 ± 4.08
	80.29 ± 2.42	**77.32 ± 2.26**	**78.02 ± 2.44**	**78.34 ± 2.31**	**78.29 ± 6.36**	**74.86 ± 3.06**

The Friedman non-parametric test is similar to ANOVA. First, for a given database, we calculate the ranking of the results obtained by each algorithm, where the best has a value of 1, and the worst has a value of k (the number of algorithms to compare). This operation is performed for the N databases considered, so finally the mean ranking is obtained using each algorithm. The null hypothesis is the assumption that the results of the algorithms are equivalent and, therefore, that they are not statistically different. Eq. (2) is the statistic used by the Friedman's test, where $R_j = \frac{1}{N}\sum_i r_i^j$ and r_i^j is the ranking for the ith algorithm and jth database. χ_F^2 is distributed as a χ^2 with $k-1$ degrees of freedom. Table 4 presents the rankings obtained.

$$\chi_F^2 = \frac{12N}{k(k+1)}\left(\sum_j R_j^2 - \frac{k(k+1)^2}{4}\right) \quad (2)$$

Table 4
Average rankings of the algorithms (Friedman).

Algorithm	Ranking
kLDEDW	2.033
CW	2.999
IB5	3.45
IB3	3.783
DE4	4.233
IB1	4.500

The p-value obtained in the test is 1.918E−6 with $\alpha = 0.05$, and therefore we can reject the null hypothesis.

Holm's post hoc method is an a posteriori procedure that allows us to compare a control algorithm (in this case kLDEDW) with the rest of the proposals. If we have k algorithms, the null hypothesis is

Table 5
Holm's procedure.

i	Algorithm	p	$\alpha/(k-i)$
1	IB1	3.282E−7	0.01
2	DE4	5.253E−6	0.0125
3	IB3	2.914E−4	0.0166
4	IB5	0.003	0.025
5	CW	0.0454	0.05

the assumption that some of the remaining $k-1$ algorithms have the same quality as the control algorithm. The procedure starts testing the hypotheses sequentially ordered according to their significance values.

$$z = (R_i - R_j) \Big/ \sqrt{\frac{k(k+1)}{6N}} \tag{3}$$

Let $p_1, p_2, \ldots, p_{k-1}$ be the sorted p-values, so that $p_1 \leqslant p_2 \leqslant \cdots \leqslant p_{k-1}$. The Holm method compares each p_i with $\alpha/(k-i)$ starting from the smallest p value. If $p_1 < \alpha/(k-1)$ then the corresponding hypothesis is rejected, allowing us to compare p_2 with $\alpha/(k-2)$. If the second hypothesis is rejected, we continue the process. If a hypothesis cannot be rejected, the rest, which have not yet been tested, are considered accepted. In Eq. (3), we can see the statistic to compare the ith algorithm with the jth. In our case, all hypotheses are rejected for $\alpha = 0.05$ (see Table 5).

Noting that the differences between the six algorithms are statistically significant and considering the results of the rankings, we can conclude that kLDEDW obtains the most satisfactory results.

5. Conclusions

In this paper, we have explored the possibility of selecting a suitable number of neighbours with feature weighting for the nearest neighbour rule. The main characteristics of our approach are twofold. First, the weights are not equal for all data but depend on the label and therefore, we are not searching for a classical weighting vector but a matrix. The second feature is that the number of neighbours is deduced from training data. The main difficulty of this approach is how to apply the methodology during the testing phase, where the label cannot be used to select the weights. The use of a weighted distance between the testing and training data in the nearest neighbour rule solves this problem. To optimize the model, a real-coded evolutionary algorithm has been used. The proposed methodology has been tested on datasets from the UCI repository. The number of datasets used and the cross-validation for each dataset allow a rigorous statistical analysis of the results. This analysis confirms that our approach outperforms five classification algorithms including two other local weighting methods. In future work, we plan to test this methodology in various real domains such as medical data and remote sensing.

References

AlSukker, A., Khushaba, R., Al-Ani, A., 2010. Optimizing the k-nn metric weights using differential evolution. In: Internat. Conf. on Multimedia Computing and Information Technology (MCIT), 2010, pp. 89–92.

Asuncion, A., Newman, D.J., 2007. UCI machine learning repository.

Chen, B., Liu, H., Chai, J., Bao, Z., 2009. Large margin feature weighting method via linear programming. IEEE Trans. Knowl. Data Eng. 21, 1475–1488.

Chen, L., Guo, G., Wang, K., 2011. Class-dependent projection based method for text categorization. Pattern Recognition Lett. 32, 1493–1501.

Demšar, J., 2006. Statistical comparisons of classifiers over multiple data sets. J. Mach. Learn. Res. 7, 1–30.

Eshelman, L.J., Schaffer, J.D., 1993. Real-coded genetic algorithms and interval-schemata. In: Whitley, D.L. (Ed.), Foundation of Genetic Algorithms 2. Morgan Kaufmann, San Mateo, CA, pp. 187–202.

Fernandez, F., Isasi, P., 2008. Local feature weighting in nearest prototype classification. IEEE Trans. Neural Networks 19, 40–53.

García, S., Herrera, F., 2008. An extension on "statistical comparisons of classifiers over multiple data sets" for all pairwise comparisons. J. Mach. Learn. Res. 9, 2677–2694.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H., 2009. The WEKA data mining software: An update. SIGKDD Explor. 11. Available from: <http://www.cs.waikato.ac.nz/ml/weka/index_citing.html>.

Li, C.H., Gondra, I., Liu, L., 2012. An efficient parallel neural network-based multi-instance learning algorithm. J. Supercomput., 1–17.

Liu, W., Chawla, S., 2011. Class confidence weighted knn algorithms for imbalanced data sets. In: Proceedings of the 15th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining – Volume Part II. Springer-Verlag, Heidelberg, Berlin, pp. 345–356.

Mohemmed, A.W., Zhang, M., 2008. Evaluation of particle swarm optimization based centroid classifier with different distance metrics. In: IEEE Congress on Evolutionary Computation'08, pp. 2929–2932.

Paredes, R., Vidal, E., 2006. Learning weighted metrics to minimize nearest-neighbor classification error. IEEE Trans. Pattern Anal. Machine Intell. 28, 1100–1110.

Park, D.C., 2009. Centroid neural network with weighted features. J. Circuits Syst. Comput. 18, 1353–1367.

Raymer, M., Punch, W., Goodman, E., Kuhn, L., Jain, A., 2000. Dimensionality reduction using genetic algorithms. IEEE Trans. Evol. Comput. 4, 164–171.

Raymer, M., Doom, T., Kuhn, L., Punch, W., 2003. Knowledge discovery in medical and biological datasets using a hybrid bayes classifier/evolutionary algorithm. IEEE Trans. Syst. Man Cybernet. Part B: Cybernet. 33, 802–813.

Shen, C., Wang, X., Yu, D., 2012. Feature weighting of support vector machines based on derivative saliency analysis and its application to financial data mining. Internat. J. Adv. Comput. Technol. 4, 199–206.

Sun, Y., 2007. Iterative relief for feature weighting: Algorithms, theories, and applications. IEEE Trans. Pattern Anal. Machine Intell. 29, 1035–1051.

Tahir, M.A., Bouridane, A., Kurugollu, F., 2007. Simultaneous feature selection and feature weighting using hybrid tabu search/k-nearest neighbor classifier. Pattern Recognition Lett. 28, 438–446.

Wettschereck, D., Aha, D.W., Mohri, T., 1997. A review and empirical evaluation of feature weighting methods for a class of lazy learning algorithms. Artif. Intell. Rev. 11, 273–314.