An evolutionary voting for *k*-nearest neighbours

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

A B S T R A C T

Keywords:
Evolutionary computation
Nearest-neigbour
Weighted voting

This work presents an evolutionary approach to modify the voting system of the *k*-nearest neighbours (kNN) rule we called EvoNN. Our approach results in a real-valued vector which provides the optimal relative con-tribution of the *k*-nearest neighbours. We compare two possible versions of our algorithm. One of them (EvoNN1) introduces a constraint on the resulted real-valued vector where the greater value is assigned to the nearest neighbour. The second version (EvoNN2) does not include any particular constraint on the order of the weights. We compare both versions with classical kNN and 4 other weighted variants of the kNN on 48 datasets of the UCI repository. Results show that EvoNN1 outperforms EvoNN2 and statistically obtains better results than the rest of the compared methods.

1. Introduction

Weighting in machine learning is a common technique for empha-sizing some characteristics of data to improve the resulting models. For example, weighting has been used to outline the importance of some particular instances (Blachnik & Duch, 2011) or features (Zhi, Fan, & Zhao, 2014), or rank a set of techniques in the context of en-sembles (Berikov, 2014). In a broad sense, Artificial Neural Networks (ANNs) and Support Vector Machines (SVMs) can be also seen as ex-amples of using weights in learning models but the *k*-nearest neigh-bours (kNN) has been the most common technique to benefit from weights (Mateos-García, García-Gutiérrez, & Riquelme-Santos, 2012).

kNN and its variants have been widely used in the literature to solve real problems. Rodger (2014) used a hybrid model to predict the demand of natural gas. The system was implemented integrating regression, fuzzy logic, nearest neighbour and neural networks, and considering several variables such as the price, operating expenses, cost to drill new wells, etc. If we focus on biological data, Park and Kim (2015) selected significant genes from microarrays by using a nearest-neighbour-based ensemble of classifiers. On the other hand, Park, Park, Jung, and Lee (2015) tackled the problem of designing recommender systems. For this purpose the authors presented Reversed CF (RCF), a fast item-based collaborative filtering algorithm which utilizes a *k*-nearest neighbour graph.

The main goal of a weighting system lies in the optimization (commonly by metaheuristics) of a set of weights in the training step to obtain the highest accuracy but trying not to overfit the resulting model. If we focus on kNN weighting methods, many proposals weighting features or instances can be found. In Raymer, Punch, Goodman, Kuhn, and Jain (2000) a weighting method to obtain an optimal set of features was provided. The set of features was selected by means of a kNN-based genetic algorithm using a bit vector to indicate if a feature was in the selection or not. In a later work, the same authors presented a hybrid evolutionary algorithm using a Bayesian discriminant function (Raymer, Doom, Kuhn, & Punch, 2003) and trying to isolate characteristics belonging to large datasets of biomedical origin. Moreover, Paredes and Vidal (2006) used different similarity functions to improve the behaviour of the kNN. In a first approximation, they considered a weight by feature and instance on training data resulting in a non-viable number of parameters in the learning process. Then, the authors proposed three types of reduction: a weight by class and feature (label dependency), a weight by prototype (prototype dependency) and a combination of the previous ones. The optimization process was carried out by descendant gradient. In the same line, Tahir, Bouridane, and Kurugollu (2007) showed an approach that was able to both select and weight features simultaneously by using tabu search. Furthermore, Mohemmed and Zhang (2008) presented a nearest-centroid-based classifier. This method calculated prototypical instances by considering arithmetic average from the training data. To classify an instance, the method calculated the distance to every prototype and then selected the nearest one. The optimization of the best centroids that minimized the classification error was carried out through particle swarm. Fernandez and Isasi (2008) also proposed a weighting system by using a prototype-based classifier. After a data normalization that was based on the position of the instances with respect to regions, the weights were iteratively calculated. More recently, AlSukker, Khushaba, and Al-Ani (2010) used differential evolution to find weights for different aspects of data. They described four approaches: feature weighting, neighbour weighting, class weighting and mixed weighting (features and classes), with the latter being the one providing the best results.

Weighting has also been applied to the vote system of the kNN. Thus, the distance-weighted k-nearest neighbour rule (WKNN) proposed by Dudani (1976) has been known for long. WKNN weights the votes of the k nearest neighbours (w_i) according to Eq. (1) where d_i is the distance of the ith nearest neighbour (being d_1 the distance of the nearest) regarding an instance to be classified. A similar version using a uniform weighting (UWKNN) has also been proposed where a weight is inversely proportional to the position reach among the neighbours (i.e., $w_i^u = 1/i$). Recently, both techniques have been explored working together as a new classifier called Dual-Weighted kNN (DWKNN) showing promising results where each weight was calculated according to Eq. (2) (Gou, Xiong, & Kuang, 2011). A later work of Gou, Du, Zhang, and Xiong (2012) provided another version of DWKNN where the calculation of the weights were different according to Eq. (3).

$$w_i^w = \begin{cases} \dfrac{(d_k - d_i)}{(d_k - d_1)} & \text{if } d_i \neq d_1 \\ 1 & \text{if } d_i = d_1 \end{cases} \tag{1}$$

$$w_i^{dw1} = w_i^w * w_i^u \tag{2}$$

$$w_i^{dw2} = \begin{cases} \dfrac{(d_k - d_i)}{(d_k - d_1)} * \dfrac{(d_k + d_1)}{(d_k + d_i)} & \text{if } d_i \neq d_1 \\ 1 & \text{if } d_i = d_1 \end{cases} \tag{3}$$

Although all the previous approaches provided improvements regarding the classical kNN performance, they have not explored the possible better suitability of evolutionary computation for the optimization of the neighbours weights. Thus, we propose an evolutionary method to improve the kNN rule by altering the vote system knowledge obtained in the training phase. We also explore the use of constraints in learning weights with two different versions of our approach and we study their reliability compared with classical kNN and other 4 weighted variants on UCI datasets (Lichman, 2013). Finally, results are statistically validated to reinforce the final conclusions.

The rest of the paper is organized as follows. Section 2 presents the elements of the two versions of the evolutionary algorithm designed to weight the vote system of the kNN. The results and several statistical tests are specified in Section 3. Finally, Section 4 presents the main findings and future work.

2. Method

In this section, two variants of our voting optimization system called *Evolutionary Voting of Nearest Neighbours* (EvoNN) are described.

2.1. Purpose and functionality

The aim of our work was to find a set of weights to modify the influence of every nearest neighbour when they voted to assign a label to an unlabelled instance. Moreover, our approach also provided a measurement about the influence of the proximity of neighbours by means of the optimized weights. Thus, the evolutionary process provides the optimal weights that have been found to improve the classification accuracy of the domain under study.

To formalize our approach we assume that the set of classes (or labels) L is represented by the natural numbers from 1 to b, with b being the number of labels. Thus, let $D = \{(e, l) \mid e \in \mathbb{R}^f$ and $l \in L = \{1, 2, \ldots, b\}\}$ be the dataset under study with f being the number of features and b the number of labels. Let *label* be a function that assigns to every element e the real class to which it belongs to. Let us suppose that D is divided in the sets TR and TS, each of them being the training set and the testing set, respectively, such that $D = TR \cup TS$ and $TR \cap TS = \emptyset$. In the training step the classification error is minimized with participation only by instances from TR. This classification error is calculated as follows. For each $x \in TR$ we compute its k nearest neighbours according to a distance function d. Let x_i with $i = 1 \ldots k$ be the neighbours to x but ranked by distance, i.e.: $d(x, x_1) \leq d(x, x_2) \ldots \leq d(x, x_k)$ and $\forall y \in TR$ with $y \neq x_i$, $d(x, x_k) < d(x, y)$. According to the standard kNN rule, the prediction of the label of x from the labels of its neighbours can be formalized:

$$predLabel(x) = \arg\max_{l \in \{1..b\}} \sum_{i=1}^{k} \delta(l, label(x_i)) \tag{4}$$

where

$$\delta(l, label(x_i)) = \begin{cases} 1 \text{ if } label(x_i) = l \\ 0 \text{ otherwise} \end{cases} \tag{5}$$

If instead of a unitary vote, we consider that the ith neighbour contributes with the weight $w_i \in \mathbb{R}$, the function (4) is redefined:

$$predLabel(x, w) = \arg\max_{l \in \{1..b\}} \sum_{i=1}^{k} \omega_i \delta(l, label(x_i)) \tag{6}$$

The function to minimize is the sum of all prediction errors of every instance x from TR. Thus, if we define the error function as

$$error(x, w) = \begin{cases} 1 \text{ if } predLabel(x, w) \neq label(x) \\ 0 \text{ otherwise} \end{cases} \tag{7}$$

then the function to optimize is:

$$\min_{w \in \mathbb{R}^k} \sum_{x \in TR} error(x, w) \tag{8}$$

With regard to the two versions of the evolutionary algorithm, they were called EvoNN1 and EvoNN2. For EvoNN1, the nearest neighbours (those with lowest distances regarding the unlabelled instance) were " heavier" and therefore, their influence (weight) had to be greater. In EvoNN2 weights had no constraints. Regarding the design of the evolutionary algorithm, it is easy to suppose that EvoNN1 and EvoNN2 were similar except on the encoding, crossover and mutation.

2.2. Voting optimization

This subsection details the search algorithm to calculate the optimum contribution of k nearest neighbours carried out by two versions of an evolutionary algorithm . It is then necessary to define their main characteristics i.e., individual encoding, genetic operators, fitness function and generational replacement policy.

2.2.1. Individual encoding

In EvoNN1 and EvoNN2, an individual was a real-valued vector with size k symbolizing the relative contribution of the k-nearest neighbours in the voting system of the kNN rule. Position 1 in the vector was associated with the nearest neighbour in distance and position k with the furthest. Moreover, a constraint was established in EvoNN1 to assure that the closest neighbours were more important i.e., $\omega_1 \geq \omega_2 \geq \ldots \omega_k$.

Regarding the initial population, both designs integrated individuals with k random values between 0 and 1 (ordered in EvoNN1). To include individuals representing classic kNN, initial population also included several vectors with the first k values set to 1 and the remaining set to 0 in the initial population e.g., $(1.0, 0.0, \ldots, 0.0)$ for

```
1:  $Fitness(W, k, TR) : error$
2:  $error = 0$
3:  for $i = 1$ to $m$ do
4:      Divide $TR$ in $s$ 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: $Evaluate(W, k, Train, Test) : error$
15: $error = 0$
16: for each $instTest$ in $Test$ do
17:     $lab = NearestN(W, k, Train, instTest)$
18:     if $lab \neq label(insTest)$ 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 $x$ in $sortedInst$ sorted by $d(x, y)$
28: end for
29: $kNeighbours = sortedInst.get(k)$
30: $labY = majorityLabel(kNeighbours, W)$
31: return $labY$
```

Fig. 1. Fitness function.

$k = 1$, $(1.0, 1.0, \ldots, 0.0)$ for $k = 2$, etc. Finally, the maximum value of 1 for a weight could be surpassed during the evolutionary process to stand out the importance of a concrete neighbour as explained in the following section.

2.2.2. *Crossover and mutation*

The goal of the crossover operator was the generation of a new individual (*offspring*) from the genotypic characteristics of a set of selected parents (two in our case, *parent*1 and *parent*2). There is also a constraint in the order of the genes in EvoNN1. Thus, the crossover operator in the *i*th gene for EvoNN1 was defined as in Eq. (9) where $BLX - \alpha$ stands for the crossover operator defined in Eshelman and Schaffer (1993) and calculated from $parent1(i)$ and $parent2(i)$, γ is a random value between 0 and 1, $max = offspring(i-1)$ and $min = minimum(parent1(i), parent2(i), offspring(i-1))$.

$$offspring(i) = \begin{cases} BLX - \alpha & \text{if } i = 1 \\ (max - min) * \gamma + min & \text{otherwise} \end{cases} \tag{9}$$

In EvoNN2, because the individuals were freely built, the crossover operator was simpler, see Eq. (10):

$$offspring(i) = \quad BLX - \alpha \quad \text{from } parent1(i) \quad \text{and } parent2(i) \tag{10}$$

Regarding the mutation operator, the *i*th gene of the individual *indiv* could change according to Eq. (11) in EvoNN1 where δ was initially a random value between 0 and 1. Then, to improve the fit through the generations, the δ value was in the interval [0, 1] during the first 10 generations. For the next 10, it was in [0, 0.9], etc. On the other hand, the mutation operator in EvoNN2 worked as can be seen in Eq. (12).

$$indiv'(i) = \begin{cases} indiv(i) + indiv(i) * \delta & \text{if } i = 1 \\ indiv(i) - indiv(i) * \delta & \text{if } i = k \\ (indiv(i-1) - indiv(i+1)) * \delta \\ \quad + indiv(i+1) & \text{otherwise} \end{cases} \tag{11}$$

$$indiv'(i) = indiv(i) \pm indiv(i) * \delta \tag{12}$$

2.2.3. *Fitness function*

EvoNN1 and EvoNN2 were defined by the same fitness function. This is because the proposed evolutionary designs were thought to minimize the classification error and therefore, the order of weights did not therefore have any influence on the fitness function. Both approaches used the $TR \subset D$ exclusively to obtain the contributions of the neighbours in the training step. Since we know the labels of the instances from *TR*, the fitness function was based on a cross-validation error rate by using kNN and the weighted voting system.

Fig. 1 shows the fitness calculation with $m \times s$ cross validations, where *m* stands for the number of iterations of the validation process (line 3) and *s* is the number of partitions of the training data *TR* (line 4). Thus, the set *TR* is randomly divided in the bags B_1, $B_2 ... B_s$ for each validation. Then, every bag B_j is evaluated through a classification process by using $TR - B_j$ as a training set. This evaluation is driven by the function *Evaluate* which we will describe later. The classification error on every B_j is accumulated on average by *partialError* (lines 7 and 9), and by *error* in every validation (line 10). Finally, the fitness value is the result of calculating the mean of all the validations (line 12).

The input parameters of the function *Evaluate* are the weighted vector *W*, the *k* value, and the subsets $TR - B_j$ and B_j (line 7).

Therefore, the result of this function is the error rate on B_j taking $TR - B_j$ as reference to calculate the neighbours.

For every single instance from the set used to measure the fitness (line 16), the returned label by the function *NearestN* is the majority one according to the k nearest instances belonging to the set used as training data (line 17). If the returned label does not correspond to the real label of the testing instance, the error is increased by 1 (line 19). Then, the resulting error is normalized with the size of the set used as testing data (line 22). Therefore, the value returned by *Evaluate* is a real number between 0 (all instances are well-classified) and 1 (all instances are misclassified).

The function *NearestN* calculates the nearest instances to the example y belonging to the set under evaluation (line 24 and seq.). Every example of the neighbours bag is then inserted in a sorted set according to the distance to y. Thus, the example at the first position will be the nearest to y and the one at the last position will be the furthest (line 27). When the algorithm selects the k nearest neighbours from the sorted set (line 29), the majority label is returned according to the relative contribution of each neighbour expressed by the vector W and according to the Eq. (6) (lines 30 and 31).

2.2.4. *Generational policy*

Regarding the transition between generations, we chose an elitist design where the best individual was transferred from one generation to the next without being affected by the mutation operator. The remaining population is built as follows: with N being the number of individuals, $C - 1$ individuals were created by cloning the best individual from the previous generation, and the next $N - C$ individuals resulted from the crossover operation. The selection of the individuals to cross was carried out by the tournament method. Furthermore, all individuals except the first one were under mutation with a probability of p.

3. Results

Applying optimization techniques to a kNN classifier is a line of research that has been widely discussed in the literature with considerable success. Finding an optimal weighting (for features or instances) increases the degrees of freedom of kNN. This allows the model to achieve a better fit in the training step and consequently improve the accuracy in testing. The novelty of this work lies in that we do not weight features or instances but the influence of each of the k nearest neighbours. In the traditional kNN model, the selected neighbours participate with a unitary vote. Our approach modifies the weight of the votes to modulate the contribution of each neighbour. Thus, our approach introduces a flexibility mechanism able to " learn" geometric arrangements of the different classes that depend on the domain under study. For example, if we focus on a dataset where the first neighbour is twice as important as the second one, and the second neighbour is in turn twice as important as the third, fourth and fifth one, we know that the traditional kNN is not able to exploit this singularity but our proposal could suggest a weighted vector (4,2,1,1,1) which would be an optimal solution.

To assess the quality of our approaches, we use 48 datasets from the repository UCI (Lichman, 2013) with different types of features and number of classes. All data were preprocessed with the same techniques i.e., binarization of nominal features, replacement of missing values and normalization to avoid the Hughes effect. Regarding the evolutionary search configuration and after a trial-and-error process, we used a population of 100 individuals, 100 generations, 10% of elitism and a mutation probability of 0.1. Regarding the parameters α (crossover), g (mutation) and k (number of neighbours) their values were set at 0.5, 20 and 5 respectively for both EvoNN1 and EvoNN2.

Although the experiments were carried out on four Intel® Xeon® Processor E7-4820, an increase of the computational cost is usually derived when using evolutionary computation. Thus, the CPU time for the optimization process can be expressed as $\frac{\#Generations \times \#Individuals \times kNN\ \text{time}}{\#threads}$ where the number of available threads depends on the workload. Taking the *German Credit Data* as an example ("credit g" in Table 1 with 1000 instances and 20 features), the evolutionary algorithm took about thirteen and a half minutes to run once.

Table 1
Comparison of accuracies of EvoNN1 and EvoNN2. In bold, the best.

Dataset	EvoNN1	EvoNN2	Dataset	EvoNN1	EvoNN2
anneal	**97.98**	97.95	heart h	**86.78**	86.04
arrhythmia	**60.13**	59.76	heart stat	**78.38**	78.13
audiology	**67.12**	65.97	hepatitis	**83.44**	81.26
australian	83.89	**84.11**	hypothyroid	**93.06**	93.04
autos	**75.42**	75.12	ionosphere	**90.45**	90.36
balance	**86.27**	86.23	iris	**99.78**	99.77
breast c.	**71.27**	69.64	kr vs kp	98.5	**98.55**
breast w.	97.15	**97.17**	labor	**81.69**	81.22
bridges v1	**70.7**	67.86	landsat	**93.86**	93.77
bridges v2	**69.79**	66.42	liver disorders	**61.49**	60.72
car	83.43	**86.89**	lung cancer	**81.4**	75.79
cmc	**63.21**	63.07	lymph	**80.87**	80.47
colic	**79.67**	79.55	mfeat factors	97.1	**97.13**
credit a	**84.34**	83.75	mfeat fourier	**86.19**	85.91
credit g	72.17	**72.81**	mfeat karhunen	**98.04**	98
cylinder b	80.19	80.19	mfeat morph.	97.24	**97.31**
dermatology	95.39	**95.62**	mfeat pixel	**96.93**	96.92
diabetes	**71.73**	71.6	mfeat zernike	92.05	92.05
ecoli	**93.17**	92.81	molecular bio.	**38.34**	32.47
flags	**55.4**	54.73	monks 1	47.39	**54.79**
glass	60.8	**62.57**	monks 2	47.12	**54.35**
haberman	**72.77**	68.49	monks 3	47.78	**46.22**
hayes	**62.35**	62.26	mushroom	100	100
heart c	**82.52**	82.14	nursery	95.91	**96.22**

In a first level, a comparison between EvoNN1 and EvoNN2 was established. Table 1 shows the results obtained. Every row represents the mean accuracy after five 10-fold cross-validations (10FCV) with different random seeds on a dataset. We provided the mean accuracy to decrease the influence of randomness in the evolutionary algorithms. The results show that EvoNN1 obtained better results for most of the datasets.

After testing the performance of both versions, we applied the Wilcoxon signed-rank test to try to assure that the differences between the two versions were statistically significant. The reason for using a non-parametric test lies in the fact that the results did not meet the necessary conditions to apply parametric tests, especially for the sphericity condition (Demšar, 2006; García & Herrera, 2008). The statistic for Wilcoxon was 290.0 and the p-Value 0.0062, so we could state that they behaved significantly different from each other.

To measure the quality of our best approach, we established a second level of comparison among IBk (implementation of kNN in the framework WEKA (Hall et al., 2009)), WKNN, UKNN, DWKNNv1 (Gou et al., 2011) and DWKNNv2 (Gou et al., 2012), and EvoNN1. All the weighting algorithms were developed from the original IBk but changing its voting system, and keeping the same k value of 5. Table 2 shows the mean accuracy obtained by the analyzed algorithms. Every dataset was evaluated for each technique as the mean of five 10FCV using five different seeds. As can be seen, the performance of our algorithm was the best in 25 out of the 48 datasets, and the second best in 7 out of the remaining 23.

Although our approach seemed to outperform the rest of competitors, the results were statistically validated again to reinforce this conclusion. Thus, we carried out a non-parametric Friedman test and a Holm post-hoc procedure to elucidate if the performance of the different algorithms was statistically different. The first step for the Friedman test is the calculation of the mean rankings reached by each technique (a ranking of 1 is the best). Table 3 shows the

Table 2
Accuracy of every studied algorithm throughout 48 datasets from UCI.

Datasets	EvoNN1	IBk	WKNN	UWKNN	DWKNN1	DWKNN2
anneal	97.98	97.1	97.85	98.43	**99.14**	98.23
arrhythmia	**60.13**	59.39	57.53	58.45	54.9	57.23
audiology	67.12	60.73	68.4	**68.78**	66.48	68.44
australian	83.89	**84.38**	82.16	83.46	80.4	82.11
autos	75.42	59.79	73.22	75.87	**76.19**	76.13
balance	86.27	**88.28**	86.81	82.65	82.01	86.81
breast c.	71.27	**73.86**	69.63	70.95	68.12	69.63
breast w.	**97.15**	97	96.09	96.36	95.39	96.04
bridges v1	**70.7**	58.34	65.11	64.93	64.08	65.24
bridges v2	**69.79**	60.47	62.75	63.37	62.15	62.75
car	83.43	**93.13**	**93.13**	88.16	88.16	**93.13**
cmc	**63.21**	45.83	45.26	45.37	44.16	44.56
colic	**79.67**	79.2	73.7	77.32	70.79	73.5
credit a	**84.34**	83.46	81.41	83.65	80.12	81.47
credit g	72.17	72.59	71.87	**72.9**	71.33	71.79
cylinder b	**80.19**	72.29	78.7	77.37	79.16	78.81
dermatology	95.39	96.26	**96.32**	95.79	95.16	**96.32**
diabetes	71.73	**74.72**	72.67	72.91	70.59	72.54
ecoli	**93.17**	86.49	82.84	83.78	80.24	82.18
flags	55.4	54.31	59.12	**60.31**	57.93	59.12
glass	60.8	66.13	**70.27**	68.05	69.49	70.07
haberman	**72.77**	71.08	70.46	67.55	64.92	69.77
hayes	62.35	27.66	67.52	67.47	**74.62**	67.52
heart c	82.52	**83.31**	80.6	80.14	76.66	80.27
heart h	**86.78**	79.41	78.56	78.56	76.67	78.54
heart stat	78.38	78.36	76.59	**79.39**	75.7	76.31
hepatitis	83.44	82.67	81.8	**83.91**	79.95	80.64
hypothyroid	93.06	**93.25**	92.62	93.13	91.2	92.5
ionosphere	**90.45**	85.61	87.46	86.33	86.89	87.62
iris	**99.78**	95.59	95.19	95.66	95.66	95.19
kr vs kp	**98.5**	96.2	95.95	95.12	93.7	95.95
labor	81.69	80.07	84.01	**85.78**	85.67	84.13
landsat	**93.86**	90.29	90.53	90.37	90.13	90.56
liver disorders	**61.49**	58.36	61.28	60.57	59.34	61.01
lung cancer	**81.4**	80.95	71.44	78.02	67.87	71.44
lymph	80.87	78.53	82.82	**84.97**	82.37	82.85
mfeat factors	**97.1**	96.56	96.69	96.51	96.08	96.68
mfeat fourier	**86.19**	81.56	81.02	81.26	80.07	80.94
mfeat karhunen	**98.04**	96.11	96.8	96.75	96.18	96.77
mfeat morph.	**97.24**	71.08	68.03	68.44	65.9	67.7
mfeat pixel	**96.93**	95.92	96.71	96.78	96.36	96.7
mfeat zernike	**92.05**	80.53	79.09	79.39	79.03	79.07
molecular bio.	**38.34**	32.57	33.56	34.76	35.24	33.56
monks 1	47.39	**52.25**	35.8	39.51	37.19	35.8
monks 2	47.12	**54.13**	38.27	40.78	38.03	38.27
monks 3	**47.78**	38.48	35.28	37.59	38.93	35.45
mushroom	100	100	100	100	100	100
nursery	95.91	**98.36**	93.63	96.1	93.23	93.63

Table 3
Mean ranking reached by every compared technique.

Technique	Ranking
EvoNN1	2.270
UWKNN	3.062
IBk	3.458
WKNN	3.594
DWKNN2	3.781
DWKNN1	4.833

Table 4
Results for Holm post-hoc procedure.

DataSet	p	z	Holm's adjusted α
DWKNNv1	1.94E−11	6.710	0.010
DWKNNv2	7.65E−5	3.955	0.013
WKNN	5.32E−4	3.464	0.017
IBk	0.002	3.109	0.025
UWKNN	0.038	2.073	0.050

rankings in our case. Then, we carried out the Friedman test. The statistic for Friedman was 48.95, distributed according to a chi-square with 5 degrees of freedom. The p-Value for Friedman was around 2.302E−9 so the null hypothesis (no statistical difference among the compared techniques) could be rejected with an $\alpha = 0.05$.

Because Friedman' test should be accompanied by a post-hoc procedure for multiple pairwise comparison, we used the Holm procedure. This procedure took our approach (EvoNN1) as the control method and compared it with each other technique by a pairwise Friedman test. The results of the procedure can be seen in Table 4 (p-Value, Friedman statistic and adjusted α for Holm procedure). In this case, every test rejected the hypothesis of no pairwise difference (p-Value below adjusted α), so we could state that our algorithm was significantly better than its competitors from a statistical point of view taking into account that it obtained the best ranking and also behaved statistically different from the rest.

From our experimentation, some facts should be outlined. First, an evolutionary algorithm to adjust the contribution of the neighbourhood improved the behaviour of the classical kNN rule. This fact could not completely be assured for the rest of weighting techniques probably due to a better adaptability of metaheuristics compared with strict functions as the suggested by DWKNN or UWKNN.

Second, the correct working of the evolutionary approach was highly correlated with the distance of the weighted neighbours to the unlabelled instances in the training phase. In other words, although EvoNN2 had more degrees of freedom in the search space, EvoNN1 reached the best performance. Although this proved true, we think this fact can be related to problems in the evolutionary search in EvoNN2 so more work could be needed to make it reach its potentially better weights.

4. Conclusions

This work presented an evolutionary method to optimize the kNN algorithm. Unlike the classical approaches that use weights on features and instances of data, we focused on adjust the contribution of each one of the k-nearest neighbours. The results showed a satisfactory performance that was statistically supported.

On the other hand, despite the use of multitasking architectures, optimization techniques and more specifically evolutionary heuristics usually consume more computational resources than lazy classifiers. Therefore it will be necessary to establish flexible and scalable parallelization pathways that are in line with the growing amount of data.

In future works, we will focus in alternative metrics to accuracy to use in unbalanced data classification. Thus, precision, recall, false positive rate, false negative rate or F-measure could be considered as the objective function to optimize.

In addition, Big Data is a clear target to research. Although there are not yet many studies about using kNN in Big Data, we will try to extrapolate our method to apply in massive datasets.

Regression problems are approachable too. The kNN algorithm works effectively when the class to predict is numeric so our method can be used in a natural way.

Finally, the strengths of our system were tested on 48 heterogeneous domains and the obtained results encourage us to implement future experiments on more specific problems. In this context, we have experience with remote sensing data (García-Gutierrez, Gonçalves-Seco, & Riquelme-Santos, 2011) and this approach could compliment the cited study in a new work.

References

AlSukker, A., Khushaba, R., & Al-Ani, A. (2010). Optimizing the k-NN metric weights using differential evolution. In *Proceedings of international conference on multimedia computing and information technology (MCIT), 2010* (pp. 89–92).

Berikov, V. (2014). Weighted ensemble of algorithms for complex data clustering. *Pattern Recognition Letters, 38*, 99–106.

Blachnik, M., & Duch, W. (2011). LVQ algorithm with instance weighting for generation of prototype-based rules. *Neural Networks, 24*(8), 824–830.(Artificial Neural Networks: Selected Papers from ICANN 2010).

Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research, 7*, 1–30.

Dudani, S. A. (1976). The distance-weighted k-nearest-neighbor rule. *IEEE Transactions on Systems, Man and Cybernetics, 6*(4), 325–327.

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

Fernandez, F., & Isasi, P. (2008). Local feature weighting in nearest prototype classification. *IEEE Transactions on Neural Networks, 19*(1), 40–53.

García-Gutierrez, J., Gonçalves-Seco, L., & Riquelme-Santos, J. C. (2011). Automatic environmental quality assessment for mixed-land zones using lidar and intelligent techniques. *Expert Systems with Applications, 38*(6), 6805–6813. http://dx.doi.org/10.1016/j.eswa.2010.12.065.

García, S., & Herrera, F. (2008). An Extension on "Statistical Comparisons of Classifiers over Multiple Data Sets" for all Pairwise Comparisons. *Journal of Machine Learning Research, 9*, 2677–2694.

Gou, J., Du, L., Zhang, Y., & Xiong, T. (2012). A new distance-weighted k-nearest neighbor classifier. *Journal of Information & Computational Science, 9*(6), 1429–1436.

Gou, J., Xiong, T., & Kuang, J. (2011). A novel weighted voting for k-nearest neighbor rule. *Journal of Computers, 6*(5), 833–840.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., & Witten, I. H. (2009). The WEKA data mining software: an update. *SIGKDD Explorations, 11*(1).

Lichman, M. (2013). UCI machine learning repository. http://archive.ics.uci.edu/ml.

Mateos-García, D., García-Gutiérrez, J., & Riquelme-Santos, J. C. (2012). On the evolutionary optimization of k-NN by label-dependent feature weighting. *Pattern Recognition Letters, 33*(16), 2232–2238. doi:10.1016/j.patrec.2012.08.011.

Mohemmed, A. W., & Zhang, M. (2008). Evaluation of particle swarm optimization based centroid classifier with different distance metrics. In *Proceedings of IEEE congress on evolutionary computation'08* (pp. 2929–2932).

Paredes, R., & Vidal, E. (2006). Learning weighted metrics to minimize nearest-neighbor classification error. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 28*(7), 1100–1110.

Park, C. H., & Kim, S. B. (2015). Sequential random k-nearest neighbor feature selection for high-dimensional data. *Expert Systems with Applications, 42*(5), 2336–2342. http://dx.doi.org/10.1016/j.eswa.2014.10.044.

Park, Y., Park, S., Jung, W., & Lee, S-g. (2015). Reversed CF: a fast collaborative filtering algorithm using a k-nearest neighbor graph. *Expert Systems with Applications, 42*(8), 4022–4028. http://dx.doi.org/10.1016/j.eswa.2015.01.001.

Raymer, M., Doom, T., Kuhn, L., & Punch, W. (2003). Knowledge discovery in medical and biological datasets using a hybrid bayes classifier/evolutionary algorithm. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, 33*(5), 802–813.

Raymer, M., Punch, W., Goodman, E., Kuhn, L., & Jain, A. (2000). Dimensionality reduction using genetic algorithms. *IEEE Transactions on Evolutionary Computation, 4*(2), 164–171.

Rodger, J. A. (2014). A fuzzy nearest neighbor neural network statistical model for predicting demand for natural gas and energy cost savings in public buildings. *Expert Systems with Applications, 41*(4, Part 2), 1813–1829. http://dx.doi.org/10.1016/j.eswa.2013.08.080.

Tahir, M. A., Bouridane, A., & Kurugollu, F. (2007). Simultaneous feature selection and feature weighting using hybrid tabu search/k-nearest neighbor classifier. *Pattern Recognition Letters, 28*(4), 438–446.

Zhi, X., Fan, J., & Zhao, F. (2014). Robust local feature weighting hard c-means clustering algorithm. *Neurocomputing, 134*, 20–29.