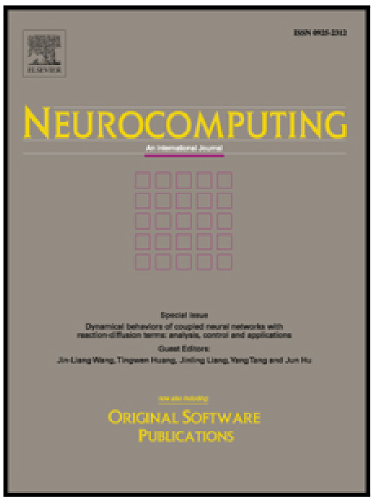
On the Evolutionary Weighting of Neighbours and Features in the k-Nearest Neighbour Rule

Article *in* Neurocomputing · September 2017

DOI: 10.1016/j.neucom.2016.08.159



On the Evolutionary Weighting of Neighbours and Features in the k-Nearest Neighbour Rule

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

PII: S0925-2312(17)31533-3
DOI: 10.1016/j.neucom.2016.08.159
Reference: NEUCOM 18896

To appear in: *Neurocomputing*

Received date: 2 December 2015
Revised date: 15 June 2016
Accepted date: 1 August 2016

On the Evolutionary Weighting of Neighbours and Features in the k-Nearest Neighbour Rule

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

Department of Computer Science, University of Seville
Avda. Reina Mercedes S/N, 41012 Seville (Spain)

Abstract

This paper presents an evolutionary method for modifying the behaviour of the k-Nearest-Neighbour classifier (kNN) called Simultaneous Weighting of Attributes and Neighbours (SWAN). Unlike other weighting methods, SWAN presents the ability of adjusting the contribution of the neighbours and the significance of the features of the data. The optimization process focuses on the search of two real-valued vectors. One of them represents the votes of neighbours, and the other one represents the weight of each feature.

The synergy between the two sets of weights found in the optimization process helps to improve significantly, the classification accuracy. The results on 35 datasets from the UCI repository suggest that SWAN statistically outperforms the other weighted kNN methods.

Keywords:
evolutionary computation, neighbours weighting, feature weighting

*Tel: +34-954 555 964; Fax: +34-954 557 139
Email addresses: mateosg@us.es (Daniel Mateos-García), jorgarcia@us.es (Jorge García-Gutiérrez), riquelme@us.es (José C. Riquelme-Santos)

1. Introduction

Weighting is a common technique used to optimize supervised learning [1, 2]. A proper fit of weights in the training step may lead to an improvement in the accuracy of a model. Artificial Neural Networks (ANNs) and Support Vector Machines (SVMs) could be the most evident examples of using weights in learning models. However, weighting is also commonly applied to other supervised learning techniques such as the k-Nearest-Neighbour (kNN) classifier [3].

Most of the proposals on weighting methods were developed for feature or instance selection (the latter have also been known as prototype selection [4]). For instance, Raymer et al. [5] performed a feature selection through a KNN-based genetic algorithm that optimised a weighting vector. In a later work, they provided an improved hybrid evolutionary algorithm which is based on the Bayesian discriminant function [6]. A similar method using tabu search was equally developed by Tahir et al. [7].In recent time, many scholars have focused on the techniques that carried out both feature and instance selection which produced better results at the expense of increasing execution time [8].

kNN can be significantly useful to weighting techniques [9]. For that reason, Paredes et al. [10] used different similarity functions optimized by weighting. A weight by each feature and instance on training data was considered, resulting in a non-viable number of parameters in the optimization process in a first approximation. Then, the authors presented three types of reduction, namely: 1) a weight by class and feature (label dependency), 2) a weight by prototype (prototype dependency) and 3) a combination of the previous ones i.e.1 and 2. The optimization process was performed by descendant gradient. Additionally, Mateos et al. [11] have recently provided an evolutionary algorithm to find a matrix of weights (a weight by feature and label) beside an optimum number of neighbours in order to better explode label-dependency. A similar idea is expressed in Won et al. [12] where the authors tried to enhance kNN performance by explicitly modelling uncertainty in the classification of each feature vector (regardless label-dependence) and the optimal number of neighbours.

Weighting has also been applied to modify neighbours votes in kNN. Hence, the distance-weighted kNN rule (WKNN) was proposed by Dudani [13] and has been known for long. WKNN weighted the votes of the k nearest neighbours (w_i) according to Eq. 1 where d_i is the distance to the i-th nearest neighbour (and d_1 to the nearest) regarding an instance to be classified. A similar version using a uniform weighting (UWKNN) was also proposed. In UWKNN, each weight was inversely proportional to the position among the neighbours (i.e., $w_i^u = 1/i$). More recently, Gou et al. [14] have also investigated both techniques working together as a new kNN version called Dual-Weighted kNN (DWKNN), where each weight was calculated according to Eq. 2. A later work [15] offered another version of DWKNN where the calculation of the weights was improved according to Eq. 3. One of the latest Works carried out on the use of new distances in kNN can be seen in Jiao et al. [16] where a class-conditional weighted distance metric was presented beside a multi-hypothesis nearest-neighbour proposal based on that metric.

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

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

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

García-Gutiérrez et al. . [17] present another point of view where the authors tried to modulate the influence of the neighbourhood by weights obtained after an evolutionary optimization. More recently, a proposal to improve the original datasets with the combination of the use of individual neighbour structures, to develop new neighbourhood representations was also shown [18]. Other researchers have worked on the locally linear reconstruction of kNN which provides a principled and k-insensitive way to determine the weights of kNN learning [19]. Lately, there has been an increase in the interest in kNN and its performance on Big Data [20]. In this novel context, weighted neighbours can play an important role as can be seen in Xia et al. [21] where a weighted model based on Map-Reduce and called Spatial-Temporal Weighted K-Nearest Neighbour (STW-KNN) was proposed to improve the short-term traffic flow forecasting.

Although, there are already a large number of literature on feature and voting system weighting (weighted distances could be included in the category of voting system weighting), yet there exists no proposal up till now (to the best of our knowledge) about a strategy to optimize both the voting system of neighbours and feature selection. Hence, the hypothesis for this work was that, regarding the issue of the improvement of kNN, ”the sum is greater tan the parts” and therefore, we proposed an evolutionary method to improve the kNN rule by optimizing the contribution of the neighbours and the importance of each feature simultaneously. Then, we statistically compared its performance with that of a classic kNN and other weighted variants tested on 35 UCI datasets [22].

The remaining part of this study is organized as follows. Section 2 presents the elements of the evolutionary algorithm designed to calculate the contribution of the k nearest neighbours and the effect of every feature. The results and a number of statistical tests are specified in Section 3. Finally, Section 4 presents the conclusions and future work.

2. Method

In this section, we describe our weighting optimization method called *Simultaneous Weighting of Attributes and Neighbours* (SWAN). The purpose of this work and how the weighting vectors from the learning process are used have been presented in subsection 2.1. While subsection 2.2 exposes the optimization algorithm in detail.

2.1. Purpose and functionality

As previously described, the aim of our work is to find a set of weights to optimize the influence of every neighbour when they vote, beside the importance of every feature. Unlike common feature weighting in the literature, ours is conditioned with a weighting voting (two vectors are optimized together, the feature weights and the weights of neighbours) fusing the synergistic ideas shown in previous work [11, 17] (although label-dependency was not included in order to avoid performance issues since a much higher number of parameters would have to be optimized).

As regards the contribution of the neighbourhood, most of the studies focus on the distance between instances. This means that the nearest neighbour instances are "heavier" than the furthest ones and therefore, their influence is greater in the final voting. However, in our case, the weights are calculated by an evolutionary algorithm regardless the distance. Obtaining a real-valued vector could transform the influence of every neighbour irrespective of the class to predict in the classification step. This means that a vote of a labelled neighbour is a real value instead of the typical value of 1. An unlabelled instance is then labelled according to Eq. 5.

To show the learning process, we assume that the set of classes (or labels) is represented by the natural numbers from 1 to b, with b being the number of the labels. Therefore, let $D = \{(e, l) \mid e \in \mathbb{R}^f \text{ and } l \in \{1, 2, ..., b\}\}$ be the dataset under study with f being the number of features and b the number of labels. Let *label* be an application that assigns to every element e, the class to which it belongs to. Let's suppose that D is divided in the sets TR and TS with each of them being the training and the testing set, respectively, so that $D = TR \cup TS$ and $TR \cap TS = \emptyset$. In this manner, the instances of TS (testing set) will be used to evaluate the fitness of SWAN and so, they are not considered for the weights calculation. As will be detailed in subsection 2.2, we obtain two vectors $v = (v_1, v_2, ..., v_k)$ and $\omega = (\omega_1, \omega_2, ..., \omega_f)$ from the instances of TR exclusively. Let H be a function that transforms every feature of an instance $x \in D$ according to a set of weights ω. To classify the instance y from TS, four steps are accomplished:

– let $x' \in TR'$ be the instances resulting from transform every $x \in TR$ according to Eq. 4

– let y' be the instance resulting from transform y according to $H(y, \omega)$

– calculate the k nearest instances to y' from TR'

– if $x'_i, i : 1..k$ is each neighbour from previous step, the assigned label to the instance y' is given by Eq. 5

$$H(x, \omega) = x' = (x_1 * \omega_1, x_2 * \omega_2, ..., x_f * \omega_f) \tag{4}$$

$$\begin{aligned} &label(y', v) = \arg \max_{l \in \{1..b\}} \sum_{i=1}^{k} v_i \delta(l, label(x'_i)) \\ &where \\ &\delta(l, label(x'_i)) = \begin{cases} 1 \text{ if } label(x'_i) = l \\ 0 \text{ otherwise} \end{cases} \end{aligned} \tag{5}$$

2.2. Evolutionary optimization

This subsection details the search algorithm to calculate the optimum contribution of every nearest neighbours and features. As mentioned above, this task is done by an evolutionary algorithm and therefore, it is necessary to define its main features i.e., individual encoding, genetic operators, fitness function and generational replacement policy.

2.2.1. Individual encoding

The population for the study constisted of a set of individuals which are represented by two real-valued vectors. The first one symbolizes the relative contribution of every neighbour in the voting stage of the kNN rule, and the second one represents the importance of every data feature (see Fig. 1). Although the weights of the votes are independent of the distances of the neighbours, yet the closest neighbour is usually the most important i.e., $v_1 \geq v_2 \geq ... \geq v_k$, and as a result, we constrained the encoding of each individual.

Regarding the initial population, the votes are k sorted values between 0 and 1. To include the classic kNN, we populate with several vectors with the first k values set to 1 and the remaining set to 0 in the initial population e.g., $(1.0, 0.0, ..., 0.0)$ for k=1, $(1.0, 1.0, ..., 0.0)$ for k=2, and so on. Note that, during the evolutionary process, the maximum value of 1 for a weight may be surpassed to highlight the importance of a concrete neighbour regarding the rest.

The vector of weights for the features does not have any constraint. It consists of random values between 0 and 1 in the initial population. The maximum value of 1 can also be exceeded. If a local minimum could be reached during the evolution process, i.e. the error function returns the same value n times, all individual, except the best are substituted by a new initial population.

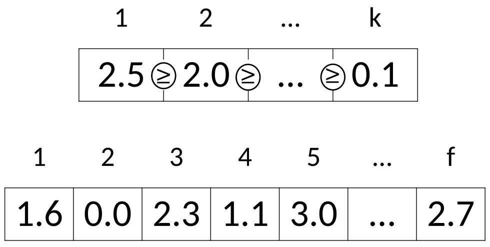
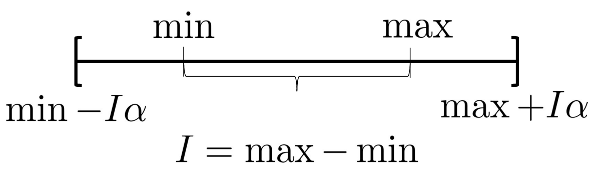


Figure 1: **Individual**

2.2.2. Crossover and mutation

After a trial-and-error procedure, we selected two concrete operators for crossover and mutation as trade-off between simplicity and quality of the results in comparison with the rest of candidates: BLX-α crossover and generation-dependent mutation.



Interval for the i-th gene of an offspring
min is the minimum value between the i-th genes of parents
max is the maximum value between the i-th genes of parents

Figure 2: **BLX-α**

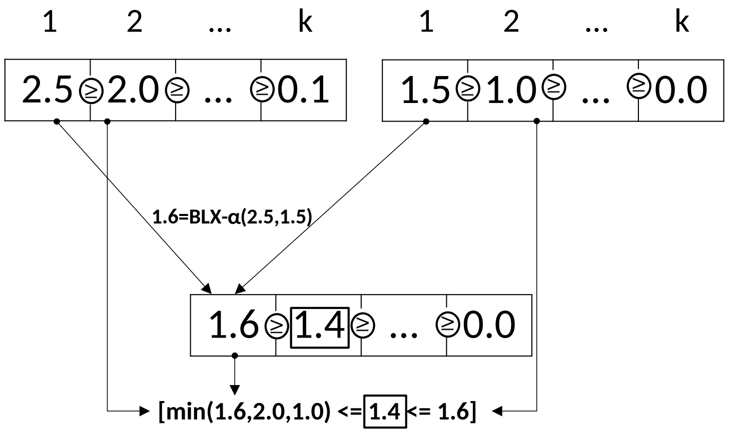


Figure 3: **Crossover of neighbours**

The main goal of the crossover operator is building a new individual ($offspring$) from the genotypic features of two parents ($parent1$ and $parent2$). Considering that there is a constraint in the order of the genes in the voting, the crossover operator for the vector of votes in the *i-th* gene has been described as follows (see Fig. 3).

$$
\begin{aligned}
&offspring(i) = \begin{cases} BLX-\alpha & \text{if } i = 1 \\ (max - min) * \gamma + min & \text{otherwise} \end{cases} \\
&where \\
&\text{- } BLX-\alpha \text{ is the crossover operator defined in Eshelman et al. [23]} \\
&\text{and calculated from } parent1(i) \text{ and } parent2(i) \\
&\text{- } \gamma \text{ is a random value between 0 and 1} \\
&\text{- } max = offspring(i-1) \\
&\text{- } min = minimum(parent1(i), parent2(i), offspring(i-1))
\end{aligned} \tag{6}
$$

The weights for the features do not have any constraint in their order, so the crossover operator can be simpler (see Fig. 2):

$$offspring(i) = BLX-\alpha \quad \text{from } parent1(i) \quad \text{and } parent2(i) \tag{7}$$

Regarding the mutation operator, the *i-th* gene of the individual could change according to Eq. 8 for the votes. δ is a random value in the interval $[0, z]$ with z being 1 initially. To find a better fit, the z value is decreased every ten generations inversely proportional to the current generation. For example, for an evolution of 100 generations, z is initially 1 and decreases by a factor of 0.1 every ten generations. Therefore, in the first ten generations $z = 1$, in the next ten $z = 0.9$, then $z = 0.8$ and so on. The idea here is to reduce the influence of the mutation operation since the individuals are closer to the end of the evolution.

This is as a result of the fact the mutation operator for feature weights is free of constraints, and its implementation is also simpler (Eq. 9).

$$indiv'(i) = \begin{cases} indiv(i) + indiv(i) * \delta & \text{if } i = 1 \\ indiv(i) - indiv(i) * \delta & \text{if } i = k \\ (indiv(i-1) - indiv(i+1)) * \delta + indiv(i+1) & \text{otherwise} \end{cases} \tag{8}$$

$$indiv'(i) = indiv(i) \pm indiv(i) * \delta \tag{9}$$

2.2.3. Error function

The evolutionary algorithm uses $TR \subset D$ exclusively to obtain the contributions of the neighbours in the training step. The fitness function is based on the cross-validation error rate by using a kNN-based classifier and the weighting vectors.

The Figure 4 shows the error calculation of $m \times n$ cross validations, where m stands for the number of iterations of the validation process (line 3) and n does for the number of partitions of training data TR (line 4). Set TR is therefore divided in subsets S_1, $S_2 ... S_n$ for each validation. Every subset S_i is evaluated through a classification process by using $TR - S_i$ as a training set. This evaluation is driven by the function evaluation which we will be

```
 1: $errorFunction(k, \omega, v, TR) : error$
 2: $error = 0$
 3: for 1 to $m$ do
 4:     Divide $TR$ randomly in $n$ subsets: $TR = S_1 \cup S_2 ... \cup S_n$
 5:     for $i = 1$ to $n$ do
 6:         $wTrain = buildModel(TR - S_i, \omega)$
 7:         $error = error + evaluate(k, wTrain, S_i, \omega, v)$
 8:     end for
 9: end for
10: $error = error/(m * n)$
11: return $error$

12: $buildModel(Train, \omega) : wTrain$
13: $wTrain = \emptyset$
14: for each $x$ in $Train$ do
15:     $x' = H(x, \omega)$ according to the Eq. 4
16:     add $x'$ to $wTrain$
17: end for
18: return $wTrain$

19: $evaluate(k, Train, Test, \omega, v) : error$
20: $error = 0$
21: for each $y$ in $Test$ do
22:     $y' = H(y, \omega)$ according to the Eq. 4
23:     $predLabel = classify(y', Train, k, v)$ according to the Eq. 5
24:     if $predLabel \neq label(y)$ then
25:         error=error+1
26:     end if
27: end for
28: $error = error/|Test|$
29: return $error$
```

Figure 4: **Error function**

described later. The classification error on every S_i is accumulated by an error in every validation (line 7). Finally, the error value is the mean of all validations (line 10).

The method *buildModel* receives two parameters: the training data and the feature weighting (line 12). It creates and returns a weighted training set with the use of Eq. 4 (lines 13 to 18). When instances are transformed, the function *evaluate* carries out the testing. The result of the *evaluate* function is the error rate on S_i taking $wTrain$ as reference to calculate the neighbours (line 7). The input parameters of *evaluate* are the number of neighbours, the transformed training data, the current testing set and the weighting vectors w and v (line 19). And so, every single instance y from the set used to measure the error (line 21), is transformed into y' according to Eq. 4 (line 22). The $classify$ function returns the majority label according to the relative contribution of each neighbour expressed by the vector v and applying Eq. 5 (line 23). If a returned label does not correspond with the true label of a testing instance, the error is increased by 1 (line 25). Finally, the resulting error is normalized according to the size of the set used as testing the data (line 28). The value returned by evaluation is then a real number between 0 (all instances are well-classified) and 1 (all instances are misclassified).

2.2.4. Generational policy

As regards the transition between generations, we chose an elitist design, where the best individual is part of the next offspring (no mutation is applied). If N is the number of individuals, the remaining population is built as follows: $C-1$ individuals are created by cloning the best individual from the previous generation. The next $N-C$ individuals result from the crossover operation. The selection of the individuals to cross is carried out by the tournament method. All individuals except the first one are affected by the mutation operator with a probability of p.

3. Results

In the experiments, we have used Java language and 35 datasets from the repository UCI [22] with different types of features and number of classes (see Table 1). All the data were preprocessed with the same techniques i.e., binarization of nominal features, replacement of missing values and normalization to avoid the Hughes effect. Through a trial-and-error process, the evolutionary algorithm was setup with a population of 100 individuals, 200 generations, 10% of elitism and a mutation probability of 0.1. Regarding the parameters α (crossover), and k (number of neighbours) their values were set at 0.5 and 5, respectively. Finally, the z value in mutation operator is decreasing at a rate of 0.1 in every generation.

To measure the precision of our approach, we established a comparison among IBk (implementation of kNN in the framework WEKA [24])), EVoN [17], WKNN, UKNN, DWKNNv1 [14] and DWKNNv2 [15]. All algorithms have been tested with k=1, k=3 and k=5, with the latter showing the best performance. Table 2 shows the mean accuracy obtained by the analyzed algorithms using 10-fold cross-validation with 5 different seeds (50 runs in the aggregate). We can see that the performance of our algorithm is the best in 16 out of the 35 datasets, and the second one in 4 out of the remaining experiments.

Additionally, as we can see in Table 1, there are imbalanced datasets. The columns "%minority" and "%majority" present the percentage of instances under the minority label and majority label respectively. However, although our evolutionary weighting methods optimize the accuracy, we can observe in Table 3 an improvement of kappa statistics too. In fact, the Pearson's r between the accuracies and kappa values is 0.81, and so, it seems that improving the global accuracy with evolutionary weighting causes an improvement of the classification performance by class.

In spite of the overall good performance of our method in direct comparison with the rest, the results must be statistically validated. For this reason, we carried out a non-parametric Friedman's test and a Holm's post-hoc procedure. The reason for using non-parametric tests lies in the high vulnerability of the necessary conditions to apply parametric tests, specially for the sphericity condition [25, 26]. The first step for the Friedman's test is the calculation of the average rankings reached by each technique (a ranking of 1 is the best). Table 4 shows the rankings. After the Friedman's test was applied, the resulting statistic was 48.58, distributed according to a chi-square with 6 degrees of freedom. The p-value for Friedman was around 9.072E-9, and so, the null hypothesis (no statistical difference among the compared techniques) could be rejected with an $\alpha = 0.05$.

The Holm's Post-hoc Procedure allows us to compare a control algorithm (in this case SWAN, the best approach candidate) with the rest avoiding problems related to family-wise error [26]. The results of the procedure can be seen in Table 5 (Friedman's statistic, p-value and adjusted α for Holm's procedure). In this case, every test rejected the hypothesis of no pairwise difference (p-values were lower than adjusted α), so we could state that our algorithm was significantly better than its competitors from a statistical point of view.

#dataset	#instances	#features	#classes	%minority	%majority
anneal	898	38	6	28	72
arrhythmia	452	279	16	0	36
audiology	226	69	24	42	58
australian	690	14	2	8	46
autos	205	25	7	34	66
balance-scale	625	4	3	0	33
breast-cancer	286	9	2	0	76
bridges_version1	105	12	6	44	56
bridges_version2	105	12	6	10	10
car	1728	6	4	42	58
cmc	1473	9	3	0	54
colic	368	22	2	10	42
credit-a	690	15	2	37	63
credit-g	1000	20	2	36	64
dermatology	366	34	6	0	64
diabetes	768	8	2	4	70
ecoli	336	7	8	10	10
flags	194	29	8	50	50
glass	214	9	7	23	43
haberman	306	3	2	10	10
hayes-roth_train	132	4	4	10	10
heart-c	303	13	5	5	31
heart-h	294	13	5	1	36
heart-statlog	270	13	2	44	56
ionosphere	351	34	2	0	54
labor	57	16	2	0	25
liver-disorders	345	6	2	10	10
lung-cancer	32	56	2	35	65
lymph	148	18	4	10	42
mfeat-karhunen	2000	64	10	1	55
mfeat-morphological	2000	6	10	10	10
mfeat-zernike	2000	47	10	26	74
monks-problems-1	124	6	2	0	92
monks-problems-2	169	6	2	9	24
monks-problems-3	122	6	2	48	52

Table 1: datasets from UCI.

data/classifier	SWAN	EVoN	kNN	DWKNN1	DWKNN2	UWKNN	WKNN
anneal	98,931	98,976	96,971	**99,065**	98,441	98,241	98,151
arrhythmia	**58,761**	58,142	58,451	53,85	55,796	56,903	55,929
audiology	**71,327**	63,186	60,708	66,726	68,142	68,584	68,23
australian	**85,478**	83,478	83,478	80,116	81,478	83,13	81,623
autos	72,098	69,659	57,756	73,171	**73,463**	72,683	71,024
balance-scale	83,232	82,592	**87,872**	81,824	86,592	82,528	86,592
breast-cancer	71,608	72,238	**73,077**	68,811	70,839	70,21	70,839
bridges_version1	59,813	59,626	58,318	60,748	**61,308**	61,121	**61,308**
bridges_version2	**63,551**	62,617	59,626	59,813	59,813	61,869	59,813
car	**95,602**	87,986	92,847	88,16	92,847	88,16	92,847
cmc	**48,364**	44,616	45,418	44,073	43,91	44,929	44,318
colic	**80,435**	79,837	79,511	72,228	75	78,261	75,109
credit-a	**85,42**	83,826	83,768	79,594	81,362	83,275	81,449
credit-g	72,4	72,8	72,76	71,04	72	**73,04**	72
dermatology	96,448	96,339	**96,612**	94,262	95,355	95,191	95,355
diabetes	73,49	73,906	**73,984**	70,417	72,474	72,708	72,396
ecoli	85,417	86,19	**86,25**	81,31	83,214	84,464	83,69
flags	55,67	54,021	53,196	55,773	57,32	**58,144**	57,32
glass	**72,617**	68,505	65,327	68,411	69,626	68,037	69,907
haberman	69,281	70,327	**70,98**	66,863	70,131	68,954	70,85
hayes-roth_train	**83,182**	52,727	26,515	76,667	68,03	67,879	68,03
heart-c	80,594	81,848	**82,112**	76,832	79,802	80,264	79,736
heart-h	**80,816**	78,707	78,707	76,327	78,571	78,503	78,707
heart-statlog	78,074	79,704	**79,926**	74,593	76,222	78,444	76,222
ionosphere	**88,604**	88,376	84,786	86,724	86,895	85,869	86,952
labor	**87,018**	83,86	81,053	86,316	84,561	85,263	84,912
liver-disorders	63,478	62,145	61,043	62,493	63,42	**63,71**	63,362
lung-cancer	73,125	73,75	**76,875**	63,75	65,625	73,75	65,625
lymph	84,73	81,081	78,378	82,027	83,378	**85,135**	83,649
mfeat-karhunen	96,85	96,77	96,14	96,36	**96,89**	96,87	96,88
mfeat-morphological	71	**71,13**	70,88	65,63	67,48	68,12	67,85
mfeat-zernike	**81,01**	80,53	80,52	78,9	78,94	79,39	78,96
monks-problems-1_train	43,548	45	**49,194**	37,258	33,387	40	33,387
monks-problems-2_train	**55,74**	51,716	53,136	36,095	36,805	38,107	36,805
monks-problems-3_train	**41,311**	37,377	40,164	30,164	30,492	32,131	30,492

Table 2: Accuracy of every studied algorithm throughout 35 datasets from UCI.

datas/classifier	SWAN	EVoN	KNN	DWKNN1	DWKNN2	UWKNN	WKNN
anneal	**1**	**1**	0,97	**1**	0,97	**1**	0,955
arrhythmia	0,383	**0,4**	0,156	0,251	0,235	0,248	0,235
audiology	**0,744**	0,684	0,707	0,665	0,642	0,738	0,666
australian	**0,638**	0,589	0,589	0,626	**0,638**	0,609	**0,638**
autos	0,662	**0,694**	0,531	0,664	0,662	**0,694**	0,662
balance-scale	0,702	0,728	**0,783**	0,704	0,742	0,717	0,742
breast-cancer	0,185	0,131	0,288	0,225	0,178	**0,385**	0,178
bridges_version1	0,414	0,329	0,292	**0,42**	**0,42**	0,329	**0,42**
bridges_version2	**0,476**	0,317	0,265	0,434	**0,476**	**0,476**	**0,476**
car	**0,878**	0,718	0,74	0,704	0,74	0,704	0,74
cmc	0,128	**0,132**	**0,132**	0,104	0,07	0,102	0,081
colic	**0,703**	0,631	0,596	0,517	0,566	0,676	0,566
credit-a	**0,762**	0,702	0,687	0,629	0,702	0,702	0,702
credit-g	0,24	0,257	0,257	0,186	0,252	0,246	**0,284**
dermatology	0,966	**0,983**	**0,983**	0,949	0,949	0,949	0,949
diabetes	**0,29**	**0,29**	**0,29**	0,202	0,242	0,225	**0,29**
ecoli	0,825	**0,846**	0,823	0,716	0,782	0,803	0,782
flags	**0,415**	0,286	0,356	0,359	0,359	0,366	0,359
glass	**0,7**	0,666	0,534	0,607	0,666	0,635	0,666
haberman	0	0	0,007	0,242	0,27	0,241	**0,323**
hayes-roth_train	**0,715**	0,41	0	0,65	0,275	0,594	0,275
heart-c	0,665	0,629	0,629	0,656	0,633	**0,696**	0,633
heart-h	**0,538**	0,4	0,4	0,4	0,454	0,486	0,486
heart-statlog	**0,662**	**0,662**	0,662	0,439	0,441	0,588	0,515
ionosphere	**0,772**	**0,772**	0,685	0,657	0,685	0,627	0,685
labor	**0,799**	0,307	0,307	0,571	0,571	0,571	0,571
liver-disorders	0,254	0,254	0,254	0,047	0,079	0,075	0,106
lung-cancer	0,086	0,3	0,086	0,086	0,3	0,3	**0,695**
lymph	0,448	0,377	0,514	0,462	0,51	**0,578**	0,51
mfeat-karhunen	0,938	0,938	0,941	0,947	0,952	**0,955**	0,952
mfeat-morphological	**0,705**	**0,705**	0,657	0,626	0,626	0,659	0,64
mfeat-zernike	0,766	**0,771**	0,768	0,757	0,763	0,768	0,763
monks-problems-1	0	**0,09**	0,029	0	0	0	0
monks-problems-2	**0,242**	0,074	0,033	0	0	0	0
monks-problems-3	0	0	0	0	0	0	0

Table 3: kappa of every studied algorithm (Cohen's kappa for binary classification and Fleiss's kappa for the multiclass case)

method	ranking
SWAN	2.314
EvoN	3.514
kNN	3.785
UWKNN	3.857
WKNN	4.285
DWKNN2	4.514
DWKNN1	5.728

Table 4: Average ranking reached by every compared technique.

dataset	**z**	**p**	**Holm's adjusted** α
DWKNNv1	6.612	3.798E-11	0.008
DWKNNv2	4.260	2.042E-5	0.010
WKNN	3.817	1.347E-4	0.012
UWKNN	2.987	0.003	0.016
kNN	2.849	0.004	0.025
EVoN	2.324	0.020	0.050

Table 5: Results for Holm's post-hoc procedure.

4. Conclusions

This work presented a method to improve the kNN rule. We unified two classic paradigms of weighting by evolutionary computation. On the one hand, we adjusted the contribution of every neighbour used in the classification step, but nevertheless, the significance of the data features was modified simultaneously in order to achieve a better result in the recognition of new instances. In spite of the complexity of the solution to optimize and the increase of the search space in comparison with single-vector based methods, the experiments showed a successful behaviour of our approach.

In future researches, we will focus on adapt evolutionary weighting algorithms, to distributed programming models such as MapReduce. By so doing, it could be possible to speed up the performance of our methods on massive data. Moreover, we will explore whether SWAN could confirm similar suitability for regression, where a change of the voting system could lead to higher changes in the model output.

The use of deep learning techniques is also a target to reach. Therefore, preprocessing training data in a hierarchical structure of multiple layers could produce good results in our research projects on specific domains, such as image recognition or natural language processing.

References

[1] E. Corchado, M. Wozniak, A. Abraham, A. de Carvalho, V. Snásel, Recent trends in intelligent data analysis., Neurocomputing 126 (2014) 1–2.

[2] A. Abraham, Special issue: Hybrid approaches for approximate reasoning., Journal of Intelligent and Fuzzy Systems 23 (2012) 41–42.

[3] D. Wettschereck, D. W. Aha, T. Mohri, A review and empirical evaluation of feature weighting methods for a class of lazy learning algorithms, Artificial Intelligence Review 11 (1997) 273–314.

[4] J. J. Valero-Mas, J. Calvo-Zaragoza, J. R. Rico-Juan, On the suitability of prototype selection methods for knn classification with distributed data, Neurocomputing In press (2016) –.

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

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

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

[8] J. Pérez-Rodríguez, A. G. A.-P. na, N. García-Pedrajas, Simultaneous instance and feature selection and weighting using evolutionary computation: Proposal and study, Applied Soft Computing 37 (2015) 416 – 443.

[9] J. Hocke, T. Martinetz, Maximum distance minimization for feature weighting, Pattern Recognition Letters 52 (2015) 48 – 52.

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

[11] D. Mateos-García, J. García-Gutiérrez, J. C. Riquelme-Santos, On the evolutionary optimization of k-nn by label-dependent feature weighting, Pattern Recognition Letters 33 (2012) 2232 – 2238.

[12] J. W. Yoon, N. Friel, Efficient model selection for probabilistic k nearest neighbour classification, Neurocomputing 149 (2015) 1098 – 1108.

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

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

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

[16] L. Jiao, Q. Pan, X. Feng, Multi-hypothesis nearest-neighbor classifier based on class-conditional weighted distance metric, Neurocomputing 151 (2015) 1468–1476.

[17] J. García-Gutiérrez, D. Mateos-García, J. C. Riquelme-Santos, Improving the k-nearest neighbour rule by an evolutionary voting approach, in: Proceedings of the 9th International Conference on Hybrid Artificial Intelligence Systems - Volume 8480, HAIS, Springer-Verlag New York, Inc., New York, NY, USA, 2014, pp. 296–305.

[18] X. Gao, T. Mu, M. Wang, Local voting based multi-view embedding, Neurocomputing 171 (2016) 901 – 909.

[19] S. kyung Lee, P. Kang, S. Cho, Probabilistic local reconstruction for k-nn regression and its application to virtual metrology in semiconductor manufacturing, Neurocomputing 131 (2014) 427 – 439.

[20] Z. Deng, X. Zhu, D. Cheng, M. Zong, S. Zhang, Efficient knn classification algorithm for big data, Neurocomputing 195 (2016) 143 – 148.

[21] D. Xia, B. Wang, H. Li, Y. Li, Z. Zhang, A distributed spatial-temporal weighted model on mapreduce for short-term traffic flow forecasting, Neurocomputing 179 (2016) 246–263.

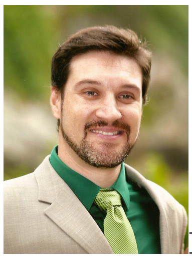
[22] M. Lichman, UCI machine learning repository, 2013. URL: `http://archive.ics.uci.edu/ml`.

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

[24] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, I. H. Witten, The WEKA data mining software: An update, SIGKDD Explorations 11 (2009).

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

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

**Daniel Mateos-Garcá** received the Ph.D. degree in Computer Engineering from the University of Seville, Spain, in 2013. Since 2003 he has been with the Department of Computer Science, University of Seville, where he is currently Assistant Professor. His primary areas of interest are machine Learning and evolutionary computation.

Jorge García Gutiérrez received the Ph.D. degree in Computer Engineering from the University of Seville, Spain, in 2012. He has been working for the Department of Computer Science of the University of Seville since 2008 where he is currently Lecturer Professor. His primary areas of interest are machine learning techniques, big data, remote sensing, data fusion and evolutionary computation.

José C. Riquelme received the M.Sc. degree in Mathematics and the Ph.D. degree in Computer Science from the University of Seville, Spain. Since 1987 he has been with the Department of Computer Science, University of Seville, where he is currently Full Professor. His primary areas

of interest are data mining, machine learning techniques and evolutionary computation.