Improving the k-Nearest Neighbour Rule by an Evolutionary Voting Approach

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

Department of Computer Science,
Avda. Reina Mercedes S/N, 41012 Seville, Spain
{jorgarcia,mateosg,riquelme}@us.es
http://www.lsi.us.es

Abstract. This work presents an evolutionary approach to modify the voting system of the k-Nearest Neighbours (kNN). The main novelty of this article lies on the optimization process of voting regardless of the distance of every neighbour. The calculated real-valued vector through the evolutionary process can be seen as the relative contribution of every neighbour to select the label of an unclassified example. We have tested our approach on 30 datasets of the UCI repository and results have been compared with those obtained from other 6 variants of the kNN predictor, resulting in a realistic improvement statistically supported.

Keywords: kNN voting, evolutionary computation, fuzzy kNN.

1 Introduction

Weighting models are common techniques in hybrid approaches [1,2] and more specifically they are usually applied to classification problems. A proper fit of weights in the training step can thus improve the accuracy of a model. Artificial Neural Networks (ANNs) and Support Vector Machines (SVMs) might be the most evident examples of using weights in learning models, although it is also usual in the k-Nearest Neighbours rule (kNN). In any case, the main goal of weighting systems is to optimize a set of weights in the training step to obtain the highest accuracy and avoid overfitting in the resulting model.

If we focus on kNN weighting methods, most proposals are based on features or instances weighting by mean of a global or local procedure. An example of global methods can be found in [3] where authors select and remove features through a kNN-based genetic algorithm. That system optimizes a weighting vector to scale the feature space and also, it uses a bit vector to select features simultaneously. In a later work, the same authors show a hybrid evolutionary algorithm based on the Bayesian discriminant function [4]. The goal of this proposal is to isolate characteristics belonging to large datasets of biomedical origin by selecting and extracting features. Other heuristics can be found in the literature. Thus, in [5] the authors present an approach that is able of both selecting and weighting features simultaneously by using tabu search.

Regarding weighted decision regions, Fernández et al. propose a local weighting system besides a prototype-based classifier [6]. After a data normalization based on the position of the instances regarding the prototype (or region) which they belong to, the weights are iteratively calculated. Alsukker et al. use differential evolution to find weights for different features of data [7]. They describe four approaches: feature weighting, neighbour weighting, class weighting and mixed weighting (features and classes), with the latter being the one providing the best results. Mohemmed et al. present a nearest-centroid-based classifier [8]. The basis of this method lies in the calculation of prototypical instances by considering the arithmetic average of the training data. When an unlabeled instance has to be classified, the distance to every prototype is calculated and the nearest one is selected. The optimization of the best centroids that minimize the classification error is carried out through particle swarm.

Moreover, Paredes et al. use different similarity functions to improve the behaviour of nearest neighbour [9]. In a first approximation they consider a weight by feature and instance on training data, resulting in a non-viable number of parameters in the learning process. Thus, the authors present 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 is carried out by descendant gradient.

Another work based on label dependency is described in [10]. This approach shows an evolutionary algorithm to find a weighted matrix (a weight by feature and class) besides an optimum number (k) of neighbours. Furthermore, the results are statistically tested beyond the classical cross-validation method. There are also references about the use of weights on unbalanced data. Liu et al. define a new measure called Class Confidence Weight (CCW) to gauge the probability that a feature value belong to a class [11]. The CCW estimation is performed by mixture models for numeric features and Bayesian nets for categorical data.

We can find another point of view in the use of weights by applying fuzzy sets theory to the kNN rule. The basis of this idea lies in the modulation of the class membership by the neighbours and the adaptation of the predictive voting system. This approach is called Fuzzy k Nearest Neighbour (Fuzzy kNN) and it presents good results in many classification problems [12]. The main handicap of this paradigm is the fuzzy membership definition, because although it can be established by the expert or even deducted from data analysis, the assignment of fuzzy values remains an open problem nowadays [13,14].

With all the previous in mind, we consider the use of a weighted system to improve the kNN rule to relativize the class membership in the training phase. Concretely, we work with the idea that the k neighbours contribute with different weights in the voting process of the kNN rule. Thus, we have designed an evolutionary system, called *Evolutionary Voting of Neighbours* (EVoN), to calculate the optimum vote weight of every neighbour from the training data and the application of the subsequent k-NN. Unlike most of the approaches in the literature our vector of weights is calculated independently of the neighbours

distance. Furthermore, its performance has been statistically validated on UCI datasets [15].

The remaining 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. The results and a number of statistical tests are specified in Section 3. And finally, Section 4 presents the conclusions and future work.

2 Method

In this section our voting optimization system called *Evolutionary Voting of Neighbours* (EVoN) is described. For this, in subsection 2.1 we present the purpose of this work and how the weighting vector from the learning process is used. The 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 modify the influence of every neighbour when they vote. Thus, we try to improve the class prediction of an unlabelled instance and therefore improve the kNN rule. Whilst there are many references of approaches that use weighting votes, as far as we know, most of the studies focus on the distance between instances. In this way, the nearest neighbours are "heavier" than the furthest ones and therefore, their influence is greater. In our case, weights are calculated by an evolutionary algorithm regardless the distance. Obtaining a real-valued vector could transform the influence of every neighbour regarding the class to predict in the classification step. This means that the vote of a labeled neighbour is a real value instead of the typical absolute value of 1. Thus, the label that classifies a new instance is the maximum of the sums of the calculated weights for the existing labels into the k nearest neighbours.

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 labels. Thus, 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 set and the testing set respectively, such 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 EVoN and therefore, they are not been considered for the weights calculation. As will be detailed in subsection 2.2, obtaining a vector $W = (\omega_1, \omega_2, ..., \omega_k)$ is carried out from the instances of TR exclusively. To classify

the instance y from TS, the k nearest instances to y are calculated from TR. If x_i is each neighbour, the assigned label to the instance y is given by:

$$label(y) = \arg \max_{l \in \{1..b\}} \sum_{i=1}^{k} \omega_i \delta(l, label(x_i)) \quad (1)$$

where $\delta(l, label(x_i))$ is 1 if $label(x_i) = l$ and 0 in other case.

2.2 Voting Optimization

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

Individual Encoding. In our approach, an individual is a real-valued vector symbolizing the relative contribution of every neighbour in the voting system of the kNN rule. In the chosen design, the value at first position is associated with the nearest neighbour, and the one at position i affects to the *i-th* neighbour. In addition, a constraint is established to ensure that the closest neighbours are more important i.e., $\omega_1 \geq \omega_2 \geq ...\omega_k$.

Regarding the initial population, it integrates individuals with k sorted values between 0 and 1. To include the classic kNN, we include 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. Finally, the maximum value of 1 for a weight may be surpassed during the evolutionary process to highlight the importance of a concrete neighbour regarding the rest.

Crossover and Mutation. As we have mentioned in subsection 2.2 there is a constraint in the order of the genes. On the other hand, the main goal of the crossover operator is building a new individual ($offspring$) from the genotypic characteristics of two parents ($parent1$ and $parent2$). To achieve both aims, the crossover operator in the *i-th* gene is defined as follows:

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

Where:

$BLX - \alpha$ is the crossover operator defined in Eshelman and Schaffer [16]
γ is a random value between 0 and 1
$max = offspring(i-1)$
$min = minimum(parent1(i), parent2(i), offspring(i-1))$

Regarding the mutation operator, if we consider the individual $indiv$, the i-th gene can change according to the following equation:

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

Where δ is a random value between 0 and 1 at the beginning. Later, the upper limit is reduced in g/G with G being the number of generations of the evolutionary algorithm and g the current generation. This reduction is used to improve the fit across generations. Thus, for $G = 100$ and $g = 10$, the δ upper limit is 1 in the first ten generations. In the following ten, it is 0.9. After another ten generations, it is 0.8 and so on.

Fitness Function. The evolutionary algorithm uses $TR \subset D$ exclusively to obtain the contributions of the neighbours in the training step. Because of we know the labels of the instances from TR, the fitness function is based on the cross-validation error rate by using kNN and the weighted voting system.

The Figure 1 shows the fitness calculation with $m \times s$ cross validations, where m is the number of iterations of the validation process (line 3) and s being the number of partitions of training data TR (line 4). Thus, the set TR is 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 average of all 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 we select the k nearest neighbours from the sorted set (line 29), the majority label is returned according

```
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 ≠ 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

to the relative contribution of each neighbour expressed by the vector W and by applying the equation 1 (lines 30 and 31).

Generational Policy. Regarding the transition between generations, we chose an elitist design where the best individual is transferred from one generation to the next but without being affected by the mutation operator. The remaining population is built as follows: being N the number of individuals, $C-1$ individuals are created by cloning the best individual from the previous generation, and 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 is affected by the mutation operation with a probability of p.

3 Results

To measure the quality of our approach, we have compared EVoN with IBk (implementation of kNN in the framework WEKA[17]) with k=1, 3 and 5. In addition, we have used an implementation of Fuzzy kNN that can be downloaded from [18].

Table 1. Accuracy of every studied algorithm

	EVoN	**IB1**	**IB3**	**IB5**	**FNN1**	**FNN3**	**FNN5**
australian	**85.6 ± 1.3**	80.2 ± 2.2	83.5 ± 1.9	84.3 ± 1.2	80.2 ± 2.2	83.8 ± 1.9	84.3 ± 1.1
balance s.	**89.6 ± 0.6**	86.8 ± 0.9	86.9 ± 1.1	88.2 ± 0.9	78.2 ± 3.5	82.3 ± 2.4	84.6 ± 1.3
breast t.	66.5 ± 5.3	68.2 ± 5.0	63.9 ± 5.9	65.3 ± 7.6	**68.5 ± 5.0**	65.9 ± 5.3	68.2 ± 5.5
breast w.	96.9 ± 1.1	95.6 ± 1.0	96.6 ± 0.9	97.1 ± 1.0	95.9 ± 1.0	96.6 ± 0.9	**97.3 ± 1.0**
car	**93.4 ± 0.5**	93.1 ± 0.5	93.1 ± 0.5	93.1 ± 0.5	76.9 ± 1.5	82.2 ± 1.5	85.9 ± 1.4
cmc	46.6 ± 1.5	44.3 ± 1.3	**47.0 ± 1.6**	45.9 ± 1.5	43.8 ± 1.3	45.4 ± 1.8	45.8 ± 1.4
diabetes	**75.1 ± 1.6**	70.9 ± 2.1	74.3 ± 2.2	74.7 ± 1.5	71.0 ± 1.8	74.2 ± 2.3	74.6 ± 1.5
ecoli	**87.1 ± 2.1**	80.2 ± 2.8	84.8 ± 2.1	86.4 ± 1.9	80.2 ± 2.8	84.8 ± 2.0	87.0 ± 1.9
glass	69.1 ± 2.8	70.0 ± 3.3	68.6 ± 3.4	66.1 ± 4.6	**70.0 ± 3.4**	68.9 ± 2.9	68.3 ± 3.5
haberman	70.9 ± 1.9	67.0 ± 2.5	**71.5 ± 2.7**	71.0 ± 1.7	66.2 ± 2.3	71.4 ± 2.3	70.5 ± 1.8
heart s.	**80.6 ± 2.5**	75.2 ± 3.1	78.5 ± 3.3	78.3 ± 3.1	75.4 ± 3.2	78.5 ± 3.3	78.4 ± 3.0
hill v.	49.0 ± 1.4	50.3 ± 1.5	51.1 ± 2.4	51.3 ± 2.5	50.2 ± 1.4	50.9 ± 2.2	**51.4 ± 2.6**
ionosphere	86.0 ± 2.4	**86.8 ± 2.4**	86.1 ± 1.7	85.6 ± 1.5	**86.8 ± 2.4**	86.1 ± 1.7	85.6 ± 1.5
liver d.	**63.2 ± 5.2**	59.3 ± 3.9	61.8 ± 3.8	58.3 ± 3.7	59.7 ± 4.0	62.1 ± 4.0	58.7 ± 3.9
lymphoma	**83.3 ± 2.7**	81.7 ± 3.0	78.7 ± 4.1	78.5 ± 4.3	82.1 ± 3.3	80.4 ± 4.0	79.4 ± 3.6
mammogr.	**82.4 ± 1.6**	76.8 ± 1.8	80.9 ± 1.8	82.2 ± 1.5	76.1 ± 2.0	80.6 ± 1.9	81.4 ± 1.4
mfeat m.	**73.2 ± 1.3**	65.8 ± 1.4	69.6 ± 1.3	71.0 ± 1.1	66.0 ± 1.6	69.7 ± 1.2	71.6 ± 1
ozone	**94.1 ± 0.2**	92.2 ± 0.9	93.9 ± 0.3	94.0 ± 0.3	92.2 ± 0.9	93.9 ± 0.3	94.0 ± 0.3
pendigits	99.3 ± 0.1	99.3 ± 0.1	99.3 ± 0.0	99.2 ± 0.0	99.3 ± 0.1	**99.4 ± 0.0**	99.2 ± 0.0
postoper.	**72.6 ± 3.7**	62.8 ± 3.0	69.2 ± 4.3	72.4 ± 3.8	56.5 ± 6.9	63.0 ± 5.6	64.9 ± 5.0
sonar	84.3 ± 3.6	84.8 ± 3.4	83.0 ± 4.9	82.5 ± 3.6	**85.2 ± 3.2**	82.3 ± 5.0	82.8 ± 3.7
sponge	88.7 ± 1.7	**92.3 ± 3.0**	88.7 ± 1.7	88.7 ± 1.7	91.7 ± 3.6	90.2 ± 2.2	88.7 ± 1.7
tae	**63.3 ± 4.8**	60.9 ± 6.1	50.3 ± 7.7	53.6 ± 5.8	62.4 ± 5.5	57.1 ± 5.2	54.4 ± 5.3
transfusion	**78.3 ± 1.3**	69.4 ± 2.0	73.8 ± 1.5	75.9 ± 1.6	68.9 ± 1.6	73.0 ± 1.3	75.6 ± 1.7
vehicle	71.3 ± 1.6	70.0 ± 1.4	70.5 ± 1.5	70.9 ± 1.5	69.8 ± 1.5	71.2 ± 1.7	**72.3 ± 1.4**
vote	93.1 ± 1.6	93.0 ± 1.5	93.9 ± 1.8	94.0 ± 2.3	93.1 ± 1.2	93.1 ± 1.7	**94.0 ± 2.3**
vowel	99.0 ± 0.3	**99.0 ± 0.3**	96.4 ± 1.4	92.7 ± 1.3	**99.0 ± 0.3**	96.4 ± 1.4	93.2 ± 1.3
wine	**96.6 ± 1.6**	94.5 ± 1.8	95.6 ± 2.2	95.4 ± 2.3	94.4 ± 1.8	95.7 ± 2.2	95.3 ± 2.3
yeast	**60.4 ± 1.1**	52.9 ± 1.4	55.2 ± 1.1	57.5 ± 1.3	53.0 ± 1.4	55.9 ± 1.3	57.6 ± 1.1
zoo	94.6 ± 2.2	95.3 ± 2.8	92.7 ± 2.7	94.6 ± 2.1	**96.2 ± 2.1**	92.7 ± 2.7	94.6 ± 2.1
	79.8 ± 2.0	**77.3 ± 2.2**	**78.0 ± 2.4**	**78.3 ± 2.3**	**76.3 ± 2.4**	**77.6 ± 2.4**	**78.0 ± 2.2**

In the experiments we have chosen 30 datasets from the repository UCI[15] with different types of features and classes. Furthermore, all data had the same preprocessing profile i.e., binarization of nominal features, normalization and replacement of missing values by the average. Regarding the evolutionary search configuration we have used a population of 100 individuals, 100 generations, 10% of elitism and a mutation probability of 0.1. In relation to the parameters α (crossover) and g (mutation) their values were 0.5 and 20 respectively. With previous parameters and using four Intel Xeon Processors E7-4820, the computation time for one execution of the evolutionary algorithm with the biggest data file (ozone dataset) with 2534 instances and 73 features was 40 minutes aproximately.

Table 1 shows the results of the analyzed algorithms for each dataset and the global averaged accuracy reached. Every dataset was evaluated with 10CV using 5 different seeds (50 executions in total). We can verify that the performance of our algorithm was the best in 16 out of the 30 datasets, and the second best in 4 out of the remaining 14. Although our approach seems to outperform the rest of competitors, the results have to be statistically validated to reinforce that conclusion. Thus, we have carried out a non-parametric Friedman test and a Holm post-hoc procedure to find out if the performances of the different algorithms are statistically different. 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 [19,20].

After applying the Friedman' test we obtain the first position in the resulting ranking of algorithms. This fact is consistent with the averaged results obtained by each algorithm. After the calculation of the Friedman statistic, a $p - value$ of $2.679E - 4$ was reached. Therefore, the null hypothesis (no statistical difference among the different algorithms) can be refused with $\alpha = 0.05$. Notice that Friedman' test is not capable of stand out the best method. Thus, the Holm post-hoc procedure allows to compare a control algorithm (in this case EVoN, the best approach candidate) with the remaining. In this case all hypothesis of equivalent performance were also rejected, so we can say that our algorithm is significantly better than its competitors from a statistical point of view.

4 Conclusions

This work presents an method able of calculating the optimum contribution of the k nearest neighbours. Unlike the classical approach that assigns an unitary vote to each neighbour, our algorithm consider a real value (a weight). The main novelty is that, to the best of our knowledge, there are not previous works that consider a distance-independent kNN voting system. Thus, we use evolutionary computation to search a weighted-vector representing the contribution of every instance from the training data. This process is carried out through the genetic operators and without the intervention of the distance function. Our voting approach was tested on 30 datasets from the UCI repository against another 6 kNN-based algorithms, with the results showing a realistic improvement that was

statistically supported. In future work, we will focus in the improvement of the fitness function of the evolutionary algorithm to avoid the full dependence with the classification error rate. The main goal of this idea is to achieve a smoothing effect of the learning curve, and therefore accomplish a more accurate searching task of solutions.

References

1. Corchado, E., Wozniak, M., Abraham, A., de Carvalho, A.C.P.L.F., Snásel, V.: Recent trends in intelligent data analysis. Neurocomputing 126, 1–2 (2014)
2. Abraham, A.: Special issue: Hybrid approaches for approximate reasoning. Journal of Intelligent and Fuzzy Systems 23(2-3), 41–42 (2012)
3. Raymer, M.L., Punch, W.F., Goodman, E.D., Kuhn, L.A., Jain, A.K.: Dimensionality reduction using genetic algorithms. IEEE Transactions on Evolutionary Computation 4(2), 164–171 (2000)
4. Raymer, M., Doom, T., Kuhn, L., Punch, W.: 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 (2003)
5. Tahir, M.A., Bouridane, A., Kurugollu, F.: Simultaneous feature selection and feature weighting using hybrid tabu search/k-nearest neighbor classifier. Pattern Recognition Letters 28(4), 438–446 (2007)
6. Fernandez, F., Isasi, P.: Local feature weighting in nearest prototype classification. IEEE Transactions on Neural Networks 19(1), 40 (2008)
7. AlSukker, A., Khushaba, R., Al-Ani, A.: Optimizing the k-nn metric weights using differential evolution. In: 2010 International Conference on Multimedia Computing and Information Technology (MCIT), pp. 89–92 (2010)
8. Mohemmed, A.W., Zhang, M.: Evaluation of particle swarm optimization based centroid classifier with different distance metrics. In: IEEE Congress on Evolutionary Computation 2008, pp. 2929–2932 (2008)
9. Paredes, R., Vidal, E.: Learning weighted metrics to minimize nearest-neighbor classification error. IEEE Transactions on Pattern Analysis and Machine Intelligence 28(7), 1100–1110 (2006)
10. Mateos-García, D., García-Gutiérrez, J., Riquelme-Santos, J.C.: On the evolutionary optimization of k-nn by label-dependent feature weighting. Pattern Recognition Letters 33(16), 2232–2238 (2012)
11. Liu, W., Chawla, S.: Class confidence weighted *k*NN algorithms for imbalanced data sets. In: Huang, J.Z., Cao, L., Srivastava, J. (eds.) PAKDD 2011, Part II. LNCS, vol. 6635, pp. 345–356. Springer, Heidelberg (2011)
12. Keller, J.M., Gray Jr., M.R.: A fuzzy k-nearest neighbor algorithm. IEEE Transactions on Systems, Man, and Cybernetics 15, 580–585 (1985)
13. Mendel, J.M.: Advances in type-2 fuzzy sets and systems. Information Sciences 177(1), 84 (2007)
14. Sanz, J., Fernández, A., Bustince, H., Herrera, F.: A genetic tuning to improve the performance of fuzzy rule-based classification systems with interval-valued fuzzy sets: Degree of ignorance and lateral position. International Journal of Approximate Reasoning 52(6), 751–766 (2011)
15. Asuncion, A., Newman, D.: UCI machine learning repository (2007)

16. Eshelman, L.J., Schaffer, J.D.: Real-coded genetic algorithms and interval-schemata. In: Whitley, D.L. (ed.) Foundation of Genetic Algorithms 2, San Mateo, CA, pp. 187–202. Morgan Kaufmann (1993)
17. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The WEKA data mining software: An update. SIGKDD Explorations 11(1) (2009)
18. Jensen, R., Shen, Q.: Computational intelligence and feature selection: Rough and fuzzy approaches (2008), http://users.aber.ac.uk/rkj/book/programs.php
19. Demšar, J.: Statistical comparisons of classifiers over multiple data sets. J. Mach. Learn. Res. 7, 1–30 (2006)
20. García, S., Herrera, F.: An Extension on "Statistical Comparisons of Classifiers over Multiple Data Sets" for all Pairwise Comparisons. Journal of Machine Learning Research 9, 2677–2694 (2008)