# A Genetic Tuned Fuzzy Classifier Based on Prototypes

**Enrique Leyva**[1] **Antonio González**[1] **Raúl Pérez**[1]

[1]Dpto de Ciencias de la Computación e IA, ETSIIT,Universidad de Granada, España
Tel.: +34 958 244019, Fax: +34 958 243317
Email: {eleyvam,A.Gonzalez,Raul_Perez}@decsai.ugr.es

**Abstract**

It is known that main drawbacks of the k-Nearest Neighbors classifier are related to the need for keeping all the training prototypes. Although there are several approaches capable to significantly reduce the size of the case base, they damage the classification accuracy. We propose a novel fuzzy approach that significantly reduces the prototypes base and also improves the classification accuracy. Its good performance is evidenced by an experimental study involving 20 prototype based classifiers and 30 databases, in which the proposal is the only approach placed among the top performers in both reduction and accuracy.

**Keywords**: Fuzzy Classifiers, Evolutionary Algorithms, Nearest Neighbor, Instance Selection

## 1. Introduction

Prototype based classifiers classify instances by comparing them with previously seen cases (prototypes). This approach has shown good performances in several domains due to the capability to learn complex target functions. They are also popular for practical applications because they tend to be easy to understand and to implement.

The most known prototype based classifier is k-Nearest Neighbors (KNN) [1]; despite its age, it remains as one of the most used classifiers and it is frequently found as benchmark in experimental studies in machine learning. Nevertheless, as KNN keeps all training instances, it has large storage requirements and needs large computational time for classification. There are several proposals to speed up KNN classification; some of them provide fast search methods to find the nearest neighbors, while others reduce the size of the case base and try to keep the classification accuracy.

The latter approach is known as instance selection (IS); it is a research field that emerged almost as soon as the prototype based classification, and remains very active until present. Despite its valuable contributions, generally those instance selection methods that achieve high reductions in the number of instances, damage the classification accuracy in some degree. Exception to this rule, are some evolutionary IS methods; nevertheless, their high computational cost puts them at a disadvantage compared to other approaches when it comes to practical application.

In this paper we present FCore, a fuzzy prototype based classifier inspired on IS approaches. This proposal has most of the attractive properties of KNN, but it has none of the main drawbacks of this classical classifier because its storage requirements are quite lower. Furthermore, in an experimental study involving 20 prototype based classifiers and 30 databases we found that FCore consistently outperformed the accuracy of KNN, and it was the only approach that achieved both high reduction and high accuracy.

Due to the success of evolutionary techniques in learning fuzzy and prototype based classifiers, we propose also an extension of FCore that uses evolutionary approaches to tune up the classifier. The resulting learner can be viewed as a two-step method having a first fast phase capable to produce a good classifier, and a second optional tuning phase capable to improve the quality of the learned classifier. For most applications the first phase should be sufficient to satisfy the requirements, but for those problems where small differences in accuracy are more critical than the computational time, the second phase can be a valuable tool.

The rest of the paper is organized as follows. Section 2 reviews some background knowledge on IS, fuzzy classification and evolutionary algorithms. Section 3 depicts the proposal. Finally, in Section 4 we expose and analyze the results of the experimental studies carried out to validate FCore.

## 2. Background

### 2.1. The IS approach for enhancing the KNN algorithm

Several authors have proposed enhancements to the KNN algorithm. We are specially interested in those from the IS field [2, 3] because they are devoted to tackle simultaneously its two main known drawbacks: the high cost of classifying new instances and the huge storage requirements.

The IS field emerged from as early as the 60s of last century. Several of the first approaches were based in a strategy known as **condensation** [4, 5, 6] which retains the class border instances and dis-

cards the internal ones, these methods tend to be extremely sensitive to noise. **Edition** [7, 8, 9] is another pioneer strategy which, in opposition to condensation, discards the instances that are harmful to the classification accuracy. This kind of methods are mainly used as noise filters and achieve small reductions in the number of instances.

After more than two decades of editing and condensing proposals, a new trend known as **hybrid** emerged combining the best of both strategies to offer noise tolerance and high reduction rates [10, 11, 12, 13]. Most of the proposals from the last two decades belong to this family.

In a radically different approach some authors have proposed methods that replace the original instances with new instances in the same vector space; this strategy is known as **prototype generation** (PG) [14, 15]. Closely related to PG is the use of **clustering** algorithms. This strategy is characterized by high reductions, and although it is generally used to complement PG, some times it is used as independent strategy [16, 17].

The above strategies tend to produce classifiers with lower classification accuracy than KNN, especially those that significantly reduce the number of training instances. On the other hand, the use of **evolutionary approaches** [18, 19] in IS has shown excellent results both in classification accuracy and database reduction. Nevertheless, as we said before, their high computational cost is a main limitation that puts them at a disadvantage compared to other approaches when it comes to practical application.

### 2.2. Fuzzy classification

In fuzzy sets theory, objects belong to a set to a given degree. Thus, given a domain $D$, a fuzzy set is a mapping in the form $D \rightarrow [0, 1]$. Applying this theory to classification, classes can be seen as fuzzy sets. Hence, fuzzy classifiers assign to input instances degrees of membership for each class rather than a single class.

This kind of labeling can be especially useful in problems where classes describe vague concepts or have imprecise borders. In such contexts, an expert system that gives degrees of membership is more effective than others giving a single class output (crisp). For example, a human expert can use the fuzzy labeling to determine whether there are several acceptable classes, a single one, or none, for taking the appropriate decision in each case. On the other hand, a crisp classifier assigns a single class even if two or more classes seem to be acceptable, and it gives no information warning the user about this situation. Furthermore, fuzzy classifiers are tolerant to the uncertainty and imprecision derived from data collection and processing, which are present in every real-world training data, even if the concepts to learn are not inherently vague.

Fuzzy sets theory has been applied by several authors to prototype based classification. Most works have been focused on designing fuzzy KNN algorithms [20, 21], or combining rough [22] and fuzzy sets theories in order to build fuzzy-rough nearest neighbor classifiers [23, 24]. All these proposals combine the advantages of fuzzy and prototype based classification, but all of them have the same drawbacks of KNN when the number of training instances is high. In fact, their scalability can be lower than that of KNN due to the use of more sophisticated and costly classification criteria.

### 2.3. Evolutionary algorithms

The evolutionary algorithms (EA) are a family of meta-heuristics inspired by the natural evolution. In them, a set (population) of candidate solutions (individuals) evolve along successive iterations (generations) by means of genetic operators.

The most common operators are mutation (random variations), selection (choice of individuals) and crossover (obtaining new individuals by mixing others). The degree of fitness or goodness of the solution represented by each individual is measured by means of a fitness function, and the probability for each individual to survive or participate in the creation of new individuals is proportional to the value of its fitness value.

EAs have been successfully used in machine learning. Specifically, they have been intensively used to build fuzzy and prototype based classifiers [19, 25]. Thanks to their great ability to explore complex solution spaces, they tend to be among the top performance methods in terms of classification accuracy.

In this paper, we will use two of the most known EAs, the classical generational and steady-state genetic algorithms. The main differences between them come from the way of changing the population among iterations. In the generational genetic algorithm (GGA) [18] the population is regenerated completely on each iteration. On the other hand, the steady-state genetic algorithm (SGA) [18] replaces one or two individuals per iteration at most.

## 3. The classifier

### 3.1. Finding prototypes

The IS method that we propose for finding prototypes, is based in the local-set concept proposed in [12]. The local set (LS) of an instance $e$ (*LS(e)*) is the set of instances whose distance to $e$ is smaller than the distance between $e$ and its nearest neighbor from a different class (nearest enemy, *ne(e)*).

In this paper, we use also two concepts related to LS: cardinality and radio. Given an instance $e$, its LS cardinality (LSC) is the number of instances in its LS ($LSC(e) = |LS(e)|$), while its LS radio (LSR) is the distance between it and its nearest enemy ($LSR(e) = dist(e, ne(e))$).

Figure 1 shows six instances in a two-dimensional space. In this example we have that $ne(A) = B$,

$LSC(A) = 3$ and $LSR(A) = r$.

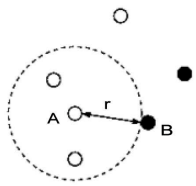


Figure 1: A LS in a two-dimensional space.

In [26], a supervised clustering algorithm based on LSs was proposed. It exploits the fact that LSs contain instances sharing the same class and spatially grouped. The algorithm (Algorithm 1) starts with a filtering (performed by the edition method *ENN* [7]) to eliminate noise and overlapping. Then it computes the LSs and sorts the instances in descendent order of their LSC. After that, each instance is tested to determine whether it is included in a previously detected cluster or selected as the representative (core) of a new cluster. The inclusion condition for an instance $e$ is that a cluster $c$ exists such that $e$ is in the LS of the core of $c$. Note that instances with higher LSC are processed first and have the best chances to be selected as cores.

**Algorithm 1** Local set based clustering (LSClustering).

**Require:** Instance set $T$
**Ensure:** $Clusters$ is a set of clusters in $T$
  $T = ENN(T)$
  $computeLocalSets(T)$
  $sortDescByLSC(T)$
  $Clusters = \{\}$
  **for** $e \in T$ **do**
    **if** $\exists(c \in Clusters)[e \in LS(c.core)]$ **then**
      $c.members = c.members \bigcup \{e\}$
    **else**
      $newC.members = \{e\}$
      $newC.core = e$
      $Clusters = Clusters \bigcup \{newC\}$
    **end if**
  **end for**

Figure 2 shows an example of how the clustering algorithm behaves. Those instances in the figure that are selected as cores have their LSs marked with dashed lines. The first cores identified are $A$ and $I$ in any order since both have the highest LSC ($LSC(A) = LSC(I) = 3$). $D$ and $E$ will result in the third and fourth cores, both with $LSC = 2$. The other instances are not cores because they are in LSs of other cores; hence, they become part of the corresponding clusters. Note that $C \in LS(A) \bigcap LS(D)$, but as $LSC(A) > LSC(D)$ the initial sorting causes that $C$ belongs to the cluster with core $A$.

From LSClustering we can get a straightforward IS method just by selecting the cluster cores as prototypes. Such method achieves high reductions

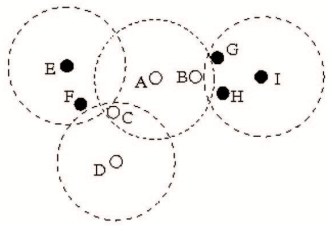


Figure 2: Using LS for clustering.

in the number of prototypes, but it loses all the information about the class borders. Hence, in those data sets having irregular and narrow borders, where border instances are decisive in classification, the classification accuracy will be poor.

Nevertheless, there is other output provided by LSClustering that can be helpful to mitigate this drawback without increasing the number of prototypes: the LSR. It provides information about the cluster borders and hence, about the class borders. In fact, given a core $c$ and its LSR, we can determine whether an instance is inside or outside of *LS(c)*. Hence, a first approach could be labeling the instances lying in *LS(c)* with the class label of $c$.

This approach produces full accuracy on non noisy instances from the training set. Nevertheless, almost never the unseen instances have exactly the same distribution than training ones. Furthermore, even in training data there are some instances noisy or lying in overlapping regions that aren't used to find prototypes and may be misclassified. These sources of uncertainty and imprecision make this approach to achieve poor classification accuracies.

As stated before, fuzzy sets theory is an effective tool to deal with uncertainty and imprecision. In the next subsection we explain how to obtain fuzzy sets from the selected prototypes in order to build a fuzzy prototype based classifier capable to be more accurate than KNN in classification.

### 3.2. From prototypes to fuzzy sets

The main idea behind a fuzzy classifier is to see classes as fuzzy sets. In prototype based classifiers, classes are described by means of prototypes. Hence, it makes sense to describe the fuzzy representation of classes by means of some fuzzy representation of prototypes.

It can be said that each crisp prototype dominates a region surrounding him, in such a way that every instance lying inside this region belongs to its class. The wideness of this region depends on the distance to the nearest prototypes, which compete with him in the classification of unseen instances.

Figure 3 shows two crisp prototypes from a hypothetical two-dimensional database. Note that the decision boundary is a line equidistant from them, and it splits the plane in such a way that instances in the left upper half will be labeled as black and those in the right lower half will be labeled as white.

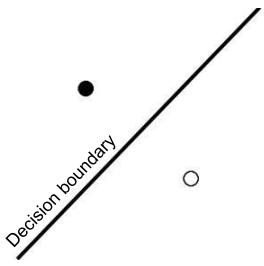


Figure 3: Two crisp prototypes in a bidimensional space.

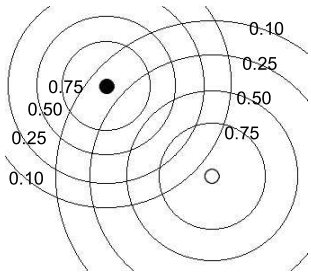


Figure 4: Two fuzzy prototypes in a bidimensional space.

A natural fuzzy extension for a prototype is a fuzzy set where the maximal membership degree corresponds to the prototype and the minimal degrees correspond to the furthest instances. The fuzzy representation of a class will be the union of all the fuzzy prototypes from this class. Thus, the membership degree of an instance $i$ to a class $C$ ($\mu_C(i)$) is the maximum membership degree of $i$ to the fuzzy representations of prototypes from $C$ as shown in Equation 1, where $\mu_p$ denotes the membership function for the fuzzy set corresponding to the prototype $p$, that is computed by equation 2.

$$\mu_C(i) = \max_{p \in prototypes(C)} (\mu_p(i)) \quad (1)$$

$$\mu_p(i) = e^{-1 * \ln 2 * \left(\frac{d(i,p)}{r_p}\right)^2} \quad (2)$$

In (2), $d(i,p)$ is the distance from $i$ to $p$, and $r_p$ is the radius of the hyperspherical region containing the 0.5-cut of the fuzzy set [1]. As reader may note, the natural value for $r_p$ is *LSR(p)*, which guarantees that every instance in *LS(p)* has at least a membership degree of 0.5 to the corresponding fuzzy set and all the others have memberships under 0.5.

Figure 4 shows two fuzzy prototypes from a hypothetical two-dimensional database. The concentric circles enclose four $\alpha$-cuts for each prototype (0.75, 0.50, 0.25 and 0.10). Note that the diameter of circles depends on the $r_p$ parameter in Equation 2, which is specific for each prototype. This way, the highest membership degree for an instance will not be necessarily to the class of the nearest prototype.

[1] The $\alpha$-cut of a fuzzy set is the crisp set of elements having membership degrees greater or equal than $\alpha$.

The classifier thus obtained (because it is fuzzy) assigns to instances a membership degree to each class instead of a single class. However, when a crisp output is required, it can be obtained in a very direct and intuitive way by assigning the class to which the instance has the greatest membership degree. This is the approach we will use in the experimental section in order to compare its performance with other classifiers.

### 3.3. The genetic tuning

The fuzzy classifier described in the previous subsection offers good results in classification accuracy using only a small portion of the training dataset as will be shown in the experimental section. Nevertheless, due to the known effectivity of EAs in the building of fuzzy and prototype based classifiers, we consider interesting to explore the possibility of using such techniques to tune up the classifier in order to improve its classification accuracy.

As it is known that EAs tend to be computationally costly, the idea is to provide a two-step classifier having a first fast phase and a second optional tuning phase. The first phase should be capable of offering good results to release the practitioners from the need to spend valuable computational time in most practical applications. On the other hand, the tuning phase must be capable of improving classification accuracy and will be reserved to problems in which there are no time constraints, or a small improvement in accuracy is desirable even at the cost of a significant increasing in time cost.

The parameter that we have tuned up using the GA is $r_p$ (see equation 2). As each prototype $p$ has its own $r_p$, there is a vector to tune which has a component for each prototype. Increasing (decreasing) $r_p$, increases (decreases) the influence of $p$ in the classification of unseen instances. We have followed an approach in which each individual encodes a candidate classifier. Thus, the chromosome is a real valued vector whose $i^{th}$ component is the value of $r_{pi}$, with $pi$ being the $i^{th}$ prototype.

The fitness function is the percent accuracy in the classification of the training set. Note that if all training instances are used to build the prototypes, the maximal fitness value is already achieved; but actually, several instances are discarded before computing the LSs because they are likely to be noisy or lying in overlapping regions. Hence, if some of these instances are not really harmful, the genetic tuning is a kind of "second opportunity" for taking them into account in the training of the classifier.

As evolutionary strategy, we have implemented the classical GGA and SGA (see section 2.3). In both cases, the crossover operator is the $BLX - \alpha$ [27] (with $\alpha = 0.5$) due to the use of real-coded chromosomes. The mutation operator modifies a single randomly selected gen, which is modified to a new value between the 50% and 150% of the current one. The initial population is built from the

classifier learned by the first step. A seed individual encoding this classifier is used to generate all the individuals in the population by modifying every gen to a new random value in the range 50%-150% of the current one in the SGA version and 50%-250% in GGA. The ranges differ because preliminary experimental results suggested us that GGA needs a greater initial diversity than SGA.

## 4. Experiments

### 4.1. Experimental setup

This experimental study includes 20 prototype based classifiers. For analytical purposes, they have been grouped into 4 families:

- The **FCore family** includes the proposals from this paper: *FCore* corresponds to running only the first step of the fuzzy classifier, whereas *FGGACore* and *FSGACore* correspond to the same classifier tuned by GGA and SGA respectively.
- The **Non IS family** includes three classifiers that retain all the training instances as knowledge base: *KNN*, *CenterNN* [28] and *KNNAdaptive* [29].
- In the **IS edition family** we included 7 edition IS methods: *AllKNN* [30], *ENN* [7], *ENNTh* [9], *ENRBF* [31], *MENN* [32], *Multiedit* [33] and *NCNEdit* [8]. They were grouped in a family apart from other IS methods because their behaviors are opposite in the sense that edition methods tend to remove small portions of the training data and to improve the classification accuracy of KNN, whereas other approaches tend to remove great portions of the training data and to damage the classification accuracy.
- The other IS methods in the study are included in the **IS non edition family**, they are *CNN* [4], *CPruner* [13], *DROP3* [11], *Explore* [34], *IB3* [10], *ICF* [12] and *PSC* [17].

In all cases we employed the implementations provided by KEEL software [35] and we assigned to the parameters (when necessary) the values proposed by their respective authors. The distance function used in the proposal was the following:

$$D = \sqrt{\sum_{a=1}^{M} d(x_a, y_a)^2} \tag{3}$$

where $d(x_a, y_a)$ is the distance for attribute $a$ and is defined as:

$$d(x_a, y_a) = \begin{cases} 1, & \text{if } x_a \text{ or } y_a \text{ is unknown} \\ VDM_a(x, y), & \text{if } a \text{ is nominal} \\ \frac{|x_a - y_a|}{(Max_a - Min_a)}, & \text{if } a \text{ is numerical} \end{cases} \tag{4}$$

where $VDM_a(x, y)$ is the Value Difference Metric [11], while $Max_a$ and $Min_a$ are maximum and minimum values for attribute $a$ respectively. Its ability to handle both numeric and nominal attributes, as well as missing values, makes it broad enough for our purposes.

| Short name | Inst | #A | #R | #I | #N | #C |
|---|---|---|---|---|---|---|
| abalone | 4174 | 8 | 7 | 0 | 1 | 28 |
| appendicitis | 106 | 7 | 7 | 0 | 0 | 2 |
| balance | 625 | 4 | 4 | 0 | 0 | 3 |
| banana | 5300 | 2 | 2 | 0 | 0 | 2 |
| breast | 286 | 9 | 0 | 0 | 9 | 2 |
| cleveland | 303 | 13 | 13 | 0 | 0 | 5 |
| contraceptive | 1473 | 9 | 0 | 9 | 0 | 3 |
| dermatology | 366 | 34 | 0 | 34 | 0 | 6 |
| ecoli | 336 | 7 | 7 | 0 | 0 | 8 |
| glass | 214 | 9 | 9 | 0 | 0 | 7 |
| haberman | 306 | 3 | 0 | 3 | 0 | 2 |
| heart | 270 | 13 | 1 | 12 | 0 | 2 |
| hepatitis | 155 | 19 | 2 | 17 | 0 | 2 |
| housevotes | 435 | 16 | 0 | 0 | 16 | 2 |
| ionosphere | 351 | 33 | 32 | 1 | 0 | 2 |
| iris | 150 | 4 | 4 | 0 | 0 | 3 |
| mammographic | 961 | 5 | 0 | 5 | 0 | 2 |
| movement_libras | 360 | 90 | 90 | 0 | 0 | 15 |
| newthyroid | 215 | 5 | 4 | 1 | 0 | 3 |
| phoneme | 5404 | 5 | 5 | 0 | 0 | 2 |
| pima | 768 | 8 | 8 | 0 | 0 | 2 |
| sonar | 208 | 60 | 60 | 0 | 0 | 2 |
| spectfheart | 267 | 44 | 0 | 44 | 0 | 2 |
| vowel | 990 | 13 | 10 | 3 | 0 | 11 |
| wdbc | 569 | 30 | 30 | 0 | 0 | 2 |
| wine | 178 | 13 | 13 | 0 | 0 | 3 |
| winequality-red | 1599 | 11 | 11 | 0 | 0 | 11 |
| winequality-white | 4898 | 11 | 11 | 0 | 0 | 11 |
| wisconsin | 699 | 9 | 0 | 9 | 0 | 2 |
| yeast | 1484 | 8 | 8 | 0 | 0 | 10 |

Table 1: Databases used in the experiments. Columns show the short name, number of attributes, real attributes, integer attributes, nominal attributes and classes in this order.

In the genetic tuning we set the population size to 20 in GGA and 50 in SGA. In both cases, the maximum number of evaluations has been 10000 and the mutation probability 0.1.

Table 1 shows the databases used in the experiments, taken from the repository KEEL [36]. All of them correspond to well-known classification problems, and as can be seen, there are among them a wide diversity in the number of instances, number and type of attributes, and number of classes. In order to facilitate replications and comparisons, the partitions used are those provided by the repository.

We ran each classifier on each database using the 10-fold cross-validation scheme. The parameters under study were the ratio of correctly classified instances on the test set (*Accuracy*) and the ratio of instances removed from the training set (*Reduction*). On the observed performances we carried out the Friedman test[2] with significance level $\alpha = 0.05$, and the null hypothesis was the absence of differences between classifiers. To find the significant differences when present, we used the post hoc Holm's test [37] for multiple comparisons between multiple methods with the same value of $\alpha$.

[2]This nonparametric test is recommended (see [37]) to be used in studies involving machine learning problems instead of the well known ANOVA, whose assumptions are most probably violated in these domains

| Classifier | Family | Accuracy | Reduction |
|---|---|---|---|
| FCore | FCore | 0.7793 | 0.9157 |
| FGGACore | FCore | 0.7810 | 0.9157 |
| FSGACore | FCore | 0.7831 | 0.9157 |
| KNN | Non IS | 0.7615 | 0.0000 |
| CenterNN | Non IS | 0.7439 | 0.0000 |
| KNNAdaptive | Non IS | 0.7869 | 0.0000 |
| AllKNN | IS edition | 0.7745 | 0.3166 |
| ENN | IS edition | 0.7766 | 0.2323 |
| ENNTh | IS edition | 0.7731 | 0.4146 |
| ENRBF | IS edition | 0.6803 | 0.3242 |
| MENN | IS edition | 0.7714 | 0.4282 |
| Multiedit | IS edition | 0.7537 | 0.3149 |
| NCNEdit | IS edition | 0.7818 | 0.2165 |
| CNN | IS non edition | 0.7306 | 0.6049 |
| CPruner | IS non edition | 0.6925 | 0.9076 |
| DROP3 | IS non edition | 0.7296 | 0.8373 |
| Explore | IS non edition | 0.7357 | 0.9810 |
| IB3 | IS non edition | 0.7140 | 0.7093 |
| ICF | IS non edition | 0.6856 | 0.7567 |
| PSC | IS non edition | 0.6719 | 0.7199 |

Table 2: Average results by classifier.

| Algorithm | Ranking | Better than | Worse than |
|---|---|---|---|
| FSGACore | 6.15 | 7 | 0 |
| KNNAdaptive | 6.43 | 7 | 0 |
| NCNEdit | 6.47 | 7 | 0 |
| ENN | 6.90 | 7 | 0 |
| FGGACore | 6.92 | 7 | 0 |
| AllKNN | 7.33 | 7 | 0 |
| ENNTh | 7.42 | 7 | 0 |
| FCore | 7.53 | 7 | 0 |
| MENN | 7.65 | 7 | 0 |
| Explore | 9.30 | 3 | 0 |
| Multiedit | 9.32 | 3 | 0 |
| KNN | 10.62 | 2 | 0 |
| CenterNN | 11.50 | 1 | 0 |
| ENRBF | 13.28 | 0 | 9 |
| CNN | 14.22 | 0 | 9 |
| DROP3 | 14.35 | 0 | 9 |
| CPruner | 14.40 | 0 | 9 |
| IB3 | 15.95 | 0 | 11 |
| ICF | 16.17 | 0 | 12 |
| PSC | 18.10 | 0 | 13 |

Table 3: Friedman test rankings and Holm's post hoc comparisons. Accuracy.

### 4.2. Results and discussion

Table 2 shows, for each classifier, the average reduction and accuracy achieved on the 30 databases. FCore, FGGACore and FSGACore are among the top performers in both parameters; furthermore, they are the only ones in this situation. Apart from them, the exceptionally high reduction achieved by Explore deserves to be mentioned; it has a great merit considering that it outperforms several methods also in accuracy. On the other hand, the highest accuracy is achieved by KNNAdaptive, but it is not as surprisingly because it uses all the training instances to classify each new one.

From the families point of view, the results are as expected. The Non IS methods achieved high accuracies due to the use of the whole database. The IS edition methods were also characterized by their high accuracies whereas their reductions were quite low. It is interesting that none member from this family outperformed KNNAdaptive in accuracy, when their main objective is to remove those instances that are supposed to be harmful to classification accuracy. It is remarkable also the poor performance of ENRBF, which is among the worst methods in accuracy despite the high portion of database that it retains. As regard to the IS non edition family, they achieved high reductions at the expense of losses in the accuracy.

According to the global averages, the FCore family outperformed the other three, being the only one that achieved high accuracies and reductions simultaneously. Within it, the tuned classifiers were slightly better in accuracy than the base FCore.

In general, FCore methods outperformed most of the others in accuracy. It is remarkable because several of the outperformed methods kept most, if not all, of the training instances while the FCore family reduced more than 91% of them. Only KNNAdaptive and NCNEdit slightly outperformed the FCore familly in accuracy. However, the first of them did not reduce the sizes of databases at all, and the other reduced them only 21.65%.

Regarding reduction, only Explore outperformed the FCore family. In turn, it was outperformed by them and other nine methods in accuracy. Very close to the FCore's reduction was that of CPruner, but its accuracy was even worse than that of Explore. All the other methods were clearly outperformed by FCore family in reduction.

Our proposal looks good in global averages; nevertheless, they are not enough to make sound conclusions. They may be biased, for example, by an extremely good (or bad) performance in a particular database. It is useful to see other perspectives such as the average rankings and to verify the soundness of observations by means of some statistical test.

Table 3 shows the classifiers sorted by their average ranking in accuracy. According to these rankings the Friedman test reveals that exist significant differences. The post hoc Holm test shows where the differences are; they are summarized in columns three and four of the table. The "better than" column shows the number of classifiers that the classifier in the row significantly outperformed whereas the column "worse than" shows the number of them that outperformed him.

Note that the first 9 classifiers (including the members of the FCore family) outperformed the same number of rivals. All IS non edition classifiers are out of this group as expected, but it is interesting that even some IS edition and Non IS classifiers are out. Such cases are: Multiedit, KNN, CenterNN and ENRBF. It is remarkable also the gap between this group and the next two classifiers (Explore and Multiedit) that comparatively outperformed less than a half of rivals.

Table 4 is equivalent to the previous one but it is focused in reduction. The Non IS classifiers were excluded from this analysis as it makes no sense for them. Also, the three FCore proposals are analyzed

| Algorithm | Ranking | Better than | Worse than |
|---|---|---|---|
| Explore | 1.07 | 11 | 0 |
| FCore | 2.57 | 10 | 0 |
| CPruner | 3.47 | 8 | 0 |
| DROP3 | 4.20 | 8 | 0 |
| ICF | 6.23 | 5 | 1 |
| IB3 | 7.00 | 5 | 2 |
| PSC | 7.10 | 5 | 2 |
| MENN | 8.33 | 2 | 4 |
| ENNTh | 9.10 | 2 | 4 |
| CNN | 9.43 | 2 | 4 |
| AllKNN | 11.00 | 0 | 7 |
| Multiedit | 11.17 | 0 | 7 |
| ENRBF | 11.57 | 0 | 7 |
| ENN | 13.83 | 0 | 10 |
| NCNEdit | 13.93 | 0 | 10 |

Table 4: Friedman test rankings and Holm's post hoc comparisons. Reduction.

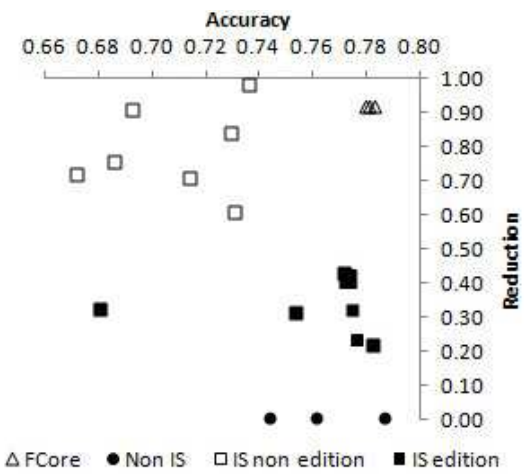


Figure 5: Accuracy vs. reduction graph.

as a single one because all of them produce the same reduction. Note that the first classifier is Explore, which is the only one outperforming 11 rivals. Very close to it is FCore, that outperformed 10. Other two methods with remarkable results are CPruner and DROP3, that complete the group of classifiers having 0 in the column "worse than".

The average rankings and statistical tests confirm that FCore classifiers are the only ones among the top performers in both accuracy and reduction. Before concluding, we will graphically show the superiority of such classifiers in Figure 5. This graph corresponds to the bi-objective problem of maximizing both accuracy and reduction. It can be said that a solution is as good as its closeness to the top-right corner. From this point of view, the FCore classifiers are the obvious winners. On the other hand, a more conservative analysis can be made from the Pareto's perspective, where a solution is said to be non dominated if there is not any other solution that outperforms it in all objectives. In such case, FSGACore is non dominated, and any Fcore classifier is non dominated if we ignore the other two. Furthermore, apart from FSGACore only Explore and KNNAdaptive are non dominated, but they are clearly biased towards one single objective.

## 5. Conclusion

We have presented FCore, a novel prototype based classifier inspired by instance selection, that uses fuzzy concepts to tackle the main drawback of such approaches: the accuracy loss. We also provided foundations to extend it to a two-step classifier having a genetic tuning phase.

In the experimental study, FCore achieved better accuracies than KNN using only a small portion of prototypes. It performed at the level of the best methods in reduction and accuracy, and it was the only one placed among the top performers in both objectives. The genetic tuning provided slight improvements in accuracies to the base method, but due to the known computational cost associated to evolutionary algorithms we recommend to use it with caution and only in those problems in which the time is not critical.

In future work we will investigate the behavior of FCore in some special kinds of problems such as imbalanced and highly noisy databases. We are also interested in exploring more sophisticated evolutionary strategies to be used in the tuning phase. With them we expect to increase the contribution of this phase to the accuracy of the classifier.

## Acknowledgements

This work has been partially funded by the Andalusian Regional Government project P09-TIC-04813 and the Spanish MEC project TIN2012-38969

## References

[1] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Trans. Inf. Theory*, 13(1):21–27, 1967.

[2] S. García, J. Derrac, J. R. Cano, and F. Herrera. Prototype selection for nearest neighbor classification: Taxonomy and empirical study. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34(3):417–435, 2012.

[3] I. Triguero, J. Derrac, S. García, and F. Herrera. A taxonomy and experimental study on prototype generation for nearest neighbor classification. *IEEE Trans. Syst., Man, Cybern., Part C*, 42(1):86–100, 2012.

[4] P. E. Hart. The condensed nearest neighbor rule. *IEEE Trans. Inf. Theory*, 14:515–516, 1968.

[5] B. V. Dasarathy. Minimal consistent set (mcs) identification for optimal nearest neighbor decision systems design. *IEEE Trans. Syst., Man, Cybern.*, 24(1):511–517, 1994.

[6] F. Angiulli. Fast nearest neighbor condensation for large data sets classification. *IEEE Trans. Knowl. Data Eng.*, 19(11):1450–1464, 2007.

[7] D. L. Wilson. Asymptotic properties of nearest neighbor rules using edited data. *IEEE Trans. Syst., Man, Cybern.*, 2(3):408–421, 1972.

[8] J.S. Sánchez, R. Barandela, A.I. Marqués, R. Alejo, and J. Badenas. Analysis of new techniques to obtain quality training sets. *Pattern Recogn. Lett.*, 24(7):1015–1022, 2003.

[9] F. Vázquez, J. S. Sánchez, and F. Pla. *A stochastic approach to wilson's editing algorithm*, volume 3523 of *LNCS*, pages 35–42. Springer-Verlag, Berlin, Heidelberg, 2005.

[10] D. W. Aha, D. Kibler, and M. K. Albert. Instance-based learning algorithms. *Mach. Learn.*, 6(1):37–66, 1991.

[11] D. R. Wilson and T. R. Martinez. Reduction techniques for instance-based learning algorithms. *Mach. Learn.*, 38:257–286, 2000.

[12] H. Brighton and C. Mellish. Advances in instance selection for instance-based learning algorithms. *Data. Min. Knowl. Discov.*, 6:153–172, 2002.

[13] K. Zhao, S. Zhou, J. Guan, and A. Zhou. C-pruner: An improved instance pruning algorithm. In *Mach. Learn. Cybern., 2003 Int. Conf.*, volume 1, pages 94–99, 2003.

[14] S. Kim and B. Oommen. Enhancing prototype reduction schemes with lvq3-type algorithms. *Pattern Recognit.*, 36(5):1083–1093, 2003.

[15] M. Lozano, J. M. Sotoca, J. S. Sánchez, F. Pla, E. Pkalska, and R. P. W. Duin. Experimental study on prototype optimisation algorithms for prototype-based classification in vector spaces. *Pattern Recognit.*, 39:1827–1838, 2006.

[16] H. A. Fayed, S. R. Hashem, and A. F. Atiya. Self-generating prototypes for pattern classification. *Pattern Recognit.*, 40(5):1498–1509, 2007.

[17] J. A. Olvera-López, J. A. Carrasco-Ochoa, and J. F. Martínez-Trinidad. A new fast prototype selection method based on clustering. *Pattern Anal. and Applicat.*, 13:131–141, 2010.

[18] J. R. Cano, F. Herrera, and M. Lozano. Using evolutionary algorithms as instance selection for data reduction in kdd: An experimental study. *IEEE Trans. Evol. Comput.*, 7(6):561–575, 2003.

[19] J. Derrac, S. García, and F. Herrera. A survey on evolutionary instance selection and generation. *International Journal of Applied Metaheuristic Computing*, 1(1):60–92, 2010.

[20] J. M. Keller, M. R. Gray, and J. A. Givens. A fuzzy k-nearest neighbor algorithm. *IEEE Trans. Syst., Man, Cybern.*, (4):580–585, 1985.

[21] J. H. Han and Y. K. Yoon. A fuzzy k-nn algorithm using weights from the variance of membership values. In *CVPR, 1999. IEEE Computer Society Conference on.*, volume 2, 1999.

[22] Z. Pawlak and A. Skowron. Rudiments of rough sets. *Information Sciences*, 177(1):3–27, 2007.

[23] M. Sarkar. Fuzzy-rough nearest neighbor algorithms in classification. *Fuzzy Sets and Systems*, 158(19):2134–2152, 2007.

[24] R. Jensen and C. Cornelis. *Fuzzy-Rough Nearest Neighbour Classification*, volume 6499 of *LNCS*, pages 56–72. Springer, 2011.

[25] A. González, R. Pérez, Y. Caises, and E. Leyva. An efficient inductive genetic learning algorithm for fuzzy relational rules. *Int. J. Comp. Intell. Syst.*, 5(2):212–230, 2012.

[26] Y. Caises, A. González, E. Leyva, and R. Pérez. Combining instance selection methods based on data characterization: An approach to increase their effectiveness. *Inf. Sci.*, 181(20):4780–4798, 2011.

[27] F. Herrera, M. Lozano, and A. M. Sánchez. A taxonomy for the crossover operator for real-coded genetic algorithms: An experimental study. *Int. J. Intell. Syst.*, 18:309–338, 2003.

[28] Q. Gao and Z. Wang. Center-based nearest neighbor classifier. *Pattern Recognit.*, 40:346–349, 2007.

[29] J. Wang, P. Neskovic, and L. N. Cooper. Improving nearest neighbor rule with a simple adaptative distance measure. *Pattern Recognit. Lett.*, 28:207–213, 2007.

[30] I. Tomek. An experiment with the edited nearest-neighbor rule. *IEEE Trans. Syst., Man, Cybern.*, 6(6):448–452, 1976.

[31] M. Grochowski and N. Jankowski. *Comparison of instance selection algorithms I. Algorithms survey*, volume 3070 of *LNCS*, pages 598–603. Springer, 2004.

[32] K. Hattori and M. Takahashi. A new edited k-nearest neighbor rule in the pattern classification problem. *Pattern Recognit.*, 33:521–528, 2000.

[33] P.A. Devijver. On the editing rate of the multiedit algorithm. *Pattern Recognit. Lett.*, 4(1):9–12, 1986.

[34] R. M. Cameron-Jones. Instance selection by encoding length heuristic with random mutation hill climbing. In *Proc. of the 8th AUS-AI 2012*, pages 99–106, Canberra, Australia, 1995.

[35] J. Alcalá-Fdez, L. Sánchez, S. García, M. J. del Jesus, S. Ventura, J. M. Garrell, J. Otero, C. Romero, J. Bacardit, V. M. Rivas, J. C. Fernández, and F. Herrera. Keel: A software tool to assess evolutionary algorithms for data mining problems. *Soft Computing*, 13(3):307–318, 2009.

[36] J. Alcalá-Fdez, A. Fernández, J. Luengo, J. Derrac, and S. García. Keel data-mining software tool: Data set repository, integration of algorithms and experimental analysis framework. *Multiple-Valued Logic and Soft Computing*, 17(2-3):255–287, 2011.

[37] J. Demsar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006.