Generating Fuzzy Partitions from Nominal and Numerical Attributes with Imprecise Values

J.M. Cadenas, M.C. Garrido, and R. Martínez

University of Murcia, Faculty of Informatic, Campus of Espinardo, 30100 Murcia, Spain
{jcadenas,carmengarrido,raquel.m.e}@um.es

Abstract. In areas of Data Mining and Soft Computing is important the discretization of numerical attributes because there are techniques that can not work with numerical domains or can get better results when working with discrete domains. The precision obtained with these techniques depends largely on the quality of the discretization performed. Moreover, in many real-world applications, data from which the discretization is carried out, are imprecise. In this paper we address both problems by proposing an algorithm to obtain a fuzzy discretization of numerical attributes from input data that show imprecise values in both numerical and nominal attributes. To evaluate the proposed algorithm we analyze the results on a set of datasets from different real-world problems.

Keywords: Fuzzy partition, Imperfect information, Fuzzy random forest ensemble, Imprecise data.

1 Introduction

The construction of fuzzy intervals in which a numerical domain is discretized supposes an important problem in the areas of data mining and soft-computing due to the determinate of these intervals can deeply affect the performance of the different classification techniques [1].

Although there are a lot of algorithms to discretization, most of them have not considered that sometimes the information available to construct the partitioning is not as precise and accurate as desirable. However, imperfect information inevitably appears in realistic domains and situations. Instrument errors or corruption from noise during experiments may give rise to information with incomplete data when measuring a specific attribute. In other cases, the extraction of exact information may be excessively costly or unfeasible. Moreover, it might be useful to complement the available data with additional information from an expert, which is usually elicited by imperfect data (interval data, fuzzy concepts, etc). In most real-world problems, data have a certain degree of imprecision. Sometimes, this imprecision is small enough for it to be safely ignored. On other occasions, the imprecision of the data can be modeled by a probability distribution. However, there is a third kind of problems, where the imprecision is significant, and a probability distribution is not the most natural way to model it. This is the case of certain practical problems where the data are inherently fuzzy [2,6,8,10].

When we have imperfect data, we have two options: the first option is to transform the original data for another kind of data which the algorithm can work; the second

K. Madani et al. (Eds.): *Computational Intelligence*, SCI 465, pp. 167–182.
DOI: 10.1007/978-3-642-35638-4_12

one is to work directly with original data without carrying out any transformation in data. When we choose the first option, we can lose information and therefore, we can lose accuracy. For this reason, it is necessary to incorporate the handling of information with attributes which may present missing and imprecise values in the discretization algorithms.

In this paper we present an algorithm, which we call EOFP (Extended Optimized Fuzzy Partitions) that obtains fuzzy partitions from imperfect information. This algorithm extends the OFP_CLASS algorithm [4] to incorporate the management of imprecise values (intervals and fuzzy values) in numerical attributes, set-valued nominal attributes (nominal attributes with imprecise values) and set-valued classes (imprecise values for the attribute class).

EOFP Algorithm follows the steps of a top-down discretization process with four iterative stages [9]: 1.- All kind of numerical values in the dataset to be discretized are ordered. 2.- The best cut point for partitioning attribute domains is found. 3.- Once the best cut point is found, the domain of each attribute is divided into two partitions. 4.- Finally, we check whether the stopping criterion is fulfilled, and if so the process is terminated.

To implement the above general discretization process, EOFP Algorithm is divided in two stages. In the first stage, we carry out a search of the best cut points for each attribute. In the second stage, based on these cut points, we use a genetic algorithm which optimizes the fuzzy sets formed from the cut points.

The structure of this study is as follows. In Section 2, we are going to present the EOFP Algorithm. In addition, in this section we are going to extend a fuzzy decision tree, which is used as base in the first stage of EOFP algorithm. This tree is able to work with imprecise information both in the values of the attributes and in the class values. Later, in Section 3, we will show various experimental results which evaluate our proposal in comparison with previously existing proposals. For these experiments we will use datasets with imprecision. In Section 4, we will show the conclusions of this study. Finally, we include Appendix A with a brief description of the combination methods used at work.

2 Designing the Algorithm

In this section we are going to present the EOFP Algorithm which is able to work with imprecise data. The EOFP Algorithm builds fuzzy partitions which guarantees for each attribute:

- Completeness (no point in the domain is outside the fuzzy partition), and
- Strong fuzzy partition (it verifies that $\forall x \in \Omega_i, \sum_{f=1}^{F_i} \mu_{B_f}(x) = 1$ where $B_1, .., B_{F_i}$ are the F_i fuzzy sets for the partition of the i numerical attribute with Ω_i domain and $\mu_{B_f}(x)$ are its functions membership).

The domain of each i numerical attribute is partitioned in trapezoidal fuzzy sets, B_1, $B_2..$, B_{F_i}, so that:

$$\mu_{B_1}(x) = \begin{cases} 1 & b_{11} \leq x \leq b_{12} \\ \frac{(b_{13}-x)}{(b_{13}-b_{12})} & b_{12} \leq x \leq b_{13} \\ 0 & b_{13} \leq x \end{cases} \quad ; \quad \mu_{B_2}(x) = \begin{cases} 0 & x \leq b_{12} \\ \frac{(x-b_{12})}{(b_{13}-b_{12})} & b_{12} \leq x \leq b_{13} \\ 1 & b_{13} \leq x \leq b_{23} \\ \frac{(b_{24}-x)}{(b_{24}-b_{23})} & b_{23} \leq x \leq b_{24} \\ 0 & b_{24} \leq x \end{cases} ;$$

$$\cdots \quad ; \quad \mu_{B_{F_i}}(x) = \begin{cases} 0 & x \leq b_{(F_i-1)3} \\ \frac{(x-b_{(F_i-1)3})}{(b_{(F_i-1)4}-b_{(F_i-1)3})} & b_{(F_i-1)3} \leq x \leq b_{(F_i-1)4} \\ 1 & b_{F_i3} \leq x \end{cases}$$

The EOFP Algorithm is composed for two stages: in the stage 1 we use a fuzzy decision tree. In this stage we get possible cut points to different attributes. In the stage 2 we carry out the process by which we optimize the cut points and make fuzzy partitions. The objective is to divide the numerical domains in fuzzy sets which will be competitive and effective to obtain a good accuracy in the classification task.

Before to describe the EOFP Algorithm, we are going to present a fuzzy decision tree witch is be able to work with imprecise data.

2.1 Fuzzy Decision Tree

In this section, we describe a fuzzy decision tree that we will use as base classifier in a Fuzzy Random Forest ensemble to evaluate fuzzy partitions generated and whose basic algorithm will be modified for the first stage of the EOFP Algorithm, as we will see later. This tree is an extension of the fuzzy decision tree that we presented in [4], to incorporate the management of imprecise values.

The tree is built from a set of examples E which are described by attributes which may be nominal expressed with crisp values and with a set of domain values (set-valued nominal attributes) and/or numerical expressed with crisp, interval and fuzzy values where there will be at least one nominal attribute which will act as class attribute. In addition, the class attribute can be expressed with a set of classes (set-valued class). Thus, the class also may be expressed in an imprecise way.

The fuzzy decision tree is based on the ID3 algorithm, where all numerical attributes have been discretized by means of a series of fuzzy sets. An initial value equal to 1 ($\chi_{root}(e_j) = 1$, where $\chi_N(e_j)$ is the membership degree of e_j to node N and e_j is j-th example from dataset) is assigned to each example e_j used in the tree learning, indicating that initially e_j is only in the root node of the tree. This value will continue to be 1 as long as the example e_j does not belong to more than one node during the tree construction process. In a classical tree, an example can only belong to one node at each moment, so its initial value (if it exists) is not modified throughout the construction process. But in a fuzzy decision tree, this value is modified when the test in a node is based on an attribute with missing, interval or fuzzy values, or with a set-valued nominal attribute.

An Attribute with Missing Values. When the example e_j has a missing value in an attribute i which is used as a test in a node N, the example descends to each child node

$N_h, h = 1, ..., H_i$ with a modified value proportionately to the weight of each child node. The modified value for each N_h is calculate as:

$$\chi_{N_h}(e_j) = \chi_N(e_j) \cdot \frac{T\chi_{N_h}}{T\chi_N}$$

where $T\chi_N$ is the sum of the weights of the examples with known value in the attribute i at node N and $T\chi_{N_h}$ is the sum of the weights of the examples with known value in the attribute i that descend to the child node N_h.

An Attribute with Interval and Fuzzy Values. When the test of a node N is based on attribute i which is numerical, each example in N modifies its weight according to the membership degree of that example to different fuzzy sets of the partition. In this case, the example e_j descends to those child nodes to which it belongs with a degree greater than 0 ($\mu_{B_f}(e_j) > 0; f = 1, ..., F_i$). Due to the characteristics of the partitions we use, the example may descend to two child nodes at most. In this case, $\chi_{N_h}(e_j) = \chi_N(e_j) \cdot \mu_{B_f}(e_j);\ \forall f|\ \mu_{B_f}(e_j) > 0;\ h = f$.

When the test of a node N is based on a numerical attribute i and the value to attribute i in e_j is a fuzzy value different from the set of partitions of the attribute, or an interval value, we need to extend the function that measures the membership degree of these type of data. This new function (denoted $\mu_{simil}(\cdot)$) captures the change in the value $\chi_N(e_j)$, when e_j descends in the fuzzy decision tree. For this reason, the membership degree of e_j is calculated using a similarity measure ($\mu_{simil}(e_j)$) between the value of attribute i in e_j and the different fuzzy sets of the partition of attribute i. Therefore, the example e_j can descend to different child nodes. In this case, $\chi_{N_h}(e_j) = \chi_N(e_j) \cdot \mu_{simil}(e_j)$.

Function $\mu_{simil}(e_j)$ is defined, for $f = 1, \ldots, F_i$, as:

$$\mu_{simil}(e_j) = \frac{\int(\min\{\mu_{e_j}(x), \mu_f(x)\})dx}{\sum_{f=1}^{F_i}\int(\min\{\mu_{e_j}(x), \mu_f(x)\})dx} \tag{1}$$

where

- $\mu_{e_j}(x)$ represents the membership function of the fuzzy or interval value of the example e_j in the attribute i.
- $\mu_f(x)$ represents the membership function of the fuzzy set of the partition of the attribute i.
- F_i is the cardinality of the partition of the attribute i.

A Set-Valued Nominal Attribute. When the test on a node of the tree is based on a nominal attribute, and some examples of that node have a set of domain values for that attribute as value, each of these examples will descend to each child node, according to a set values, with a weight proportional to the weight of each value in the set. So, if in the example e_j of node N, the nominal attribute x, with domain $\{X_1, X_2, \ldots, X_t\}$, has the value $(P_1\ X_1, P_2\ X_2, \ldots, P_t\ X_t)$ and x is the test in the node, e_j descends to each child node $N_h,\ h = 1, \ldots, t$ with weight $\chi_{N_h}(e_j) = \chi_N(e_j) \cdot P_h$.

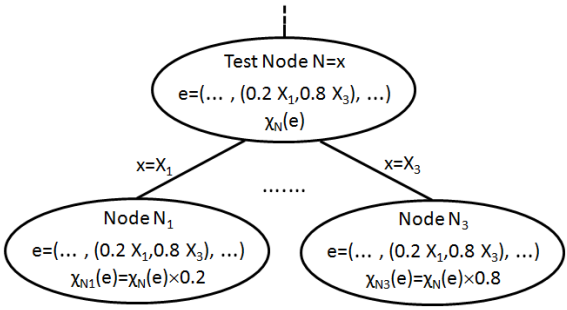

Fig. 1. Management of set-valued nominal attributes

For example, let x be a nominal attribute with domain $\{X_1, X_2, X_3\}$. The value of x in an example e_j can be expressed by $(0.2\ X_1, 0.8\ X_3)$ indicating that there are uncertainty in the value and we assign a certainty degree of 0.8 to $x = X_3$ and a certainty degree of 0.2 to $x = X_1$. Figure 1 shows, in an illustrative way, the management of this example in the fuzzy decision tree.

In a general way, we can say that the $\chi_N(e_j)$ value indicates the degree with which the example fulfills the conditions that lead to node N on the tree.

Calculating the Information Gain in an Extended Fuzzy Decision Tree. Another aspect of this extended fuzzy decision tree is the way to calculate the information gain when node N (node which is being explored at any given moment) is divided using the attribute i as test attribute. This information gain G_i^N is defined as:

$$G_i^N = I^N - I^{S_{V_i}^N} \tag{2}$$

where:

- I^N: Standard information associated with node N. This information is calculated as follows:
  1. For each class $k = 1, ..., |C|$, the value P_k^N, which is the number of examples in node N belonging to class k, is calculated:

$$P_k^N = \sum_{j=1}^{|E|} \chi_N(e_j) \cdot \mu_k(e_j) \tag{3}$$

  where $\chi_N(e_j)$ is the membership degree of example e_j to node N and $\mu_k(e_j)$ is the membership degree of example e_j to class k.
  2. P^N, which is the total number of examples in node N, is calculated.

$$P^N = \sum_{k=1}^{|C|} P_k^N$$

  3. Standard information is calculated as: $I^N = -\sum_{k=1}^{|C|} \frac{P_k^N}{P^N} \cdot \log \frac{P_k^N}{P^N}$

– $I^{S^N_{V_i}}$ is the product of three factors and represents standard information obtained by dividing node N using attribute i adjusted to the existence of missing values in this attribute.

$$I^{S^N_{V_i}} = I_1^{S^N_{V_i}} \cdot I_2^{S^N_{V_i}} \cdot I_3^{S^N_{V_i}}$$

where:

- $I_1^{S^N_{V_i}}$ = 1 - $\frac{P^{N_{m_i}}}{P^N}$, where $P^{N_{m_i}}$ is the weight of the examples in node N with missing value in attribute i.
- $I_2^{S^N_{V_i}} = \frac{1}{\sum_{h=1}^{H_i} P^{N_h}}$, H_i being the number of descendants associated with node N when we divide this node by attribute i and P^{N_h} the weight of the examples associated with each one of the descendants.
- $I_3^{S^N_{V_i}} = \sum_{h=1}^{H_i} P^{N_h} \cdot I^{N_h}$, I^{N_h} being the standard information of each descendant h of node N.

On the other hand the stopping criterion is the same that we described in [4] which is defined by the first condition reached out of the following: (a) pure node, (b) there aren't any more attributes to select, (c) reaching the minimum number of examples allowed in a node. Besides, it must be pointed out, that once an attribute has been selected as a node test, this attribute will not be selected again due to the fact that all the attributes are nominal or are partitioned.

Having constructed the fuzzy decision tree, we use it to infer an unknown class of a new example. The inference process is as follow:

Given the example e to be classified with the initial value, for instance, $\chi_{root}(e) = 1$, go through the tree from the root node. After obtain the leaf set reached by e. For each leaf reached by e, calculate the support for each class. The support for a class on a given leaf N is obtained according to the expression (3). Finally, obtain the tree's decision, c, from the information provided by the leaf set reached and the value χ with which example e activates each one of the leaves reached.

With the fuzzy decision tree presented at the moments, we have incorporated numerical attributes with imprecise values which are described by an interval or fuzzy values. Also, we have incorporated nominal attributes expressed by a set of values. In the next subsection we consider the modifications which are necessary to carry out in the phases of learning and classification to incorporate the treatment of examples whose class attribute is set-valued.

Evaluating Data with Set-Valued Classes. In the previous section, we have said that the initial weight of one example e may be equal to 1 ($\chi_{root}(e) = 1$) but this value depends on if the example has a single class or it has a set-valued class. In the first case, if the example e has a single class, the initial weight is 1 and in the second case the initial weight will depend on the number of classes that example has. Therefore, if the example e has a set-valued class with $n_{classes}$ classes, the example will be replicated $n_{classes}$ times and each replicate of the example e will have associated the weight $1/n_{classes}$.

In this case, when we perform a test of the tree to classify a dataset with set-valued classes, we can follow the decision process:

If class(e)==class$_{tree}$(e)$\wedge$size(class(e))==1 then successes++
else
If class(e)$\cap$ class$_{tree}$(e)$\neq \emptyset$ then success_or_error++ else errors++

where $class_{tree}$ is the class that fuzzy decision tree provides as output and $class(e)$ is the class value of the example e.

As result of this test, we obtain the interval $[min\_error, max\_error]$ where min_error is calculated considering only errors indicated in the variable errors from the previous process and max_error is calculated considering as errors $errors + success\_or\_error$.

With this way to classify, the tree receives an imprecise input and its output is imprecise too, because it's not possible to determine exactly a unique error.

One, we have described the fuzzy decision tree that we will use to classify, and that with some modifications, we will use in the stage 1 of the discretization algorithm, we are going to expose such algorithm. As we said earlier, the discretization algorithm EOFP is composed by two stages which we are going to present.

2.2 First Stage: Searching for Cut Points

In this stage, a fuzzy decision tree is constructed whose basic process is that described in Subsection 2.1, except that now a procedure based on priority tails is added and there are attributes that have not been discretized. To discretize these attributes, the first step is look for the cut points which will be the border between different partitions. In previous section, we are expose that to discretize attributes, we must order the values. If all data are not crisp, we need a function to order crisp, fuzzy and interval values. To order data, we use the same function that to search for the possible cut points.

To deal with non-discretized attributes, the algorithm follows the basic process in C4.5. The thresholds selected in each node of the tree for these attributes will be the split points that delimit the intervals. Thus, the algorithm that constitutes this first stage is based on a fuzzy decision tree that allows nominal attributes, numerical attributes discretized by means of a fuzzy partition, non-discretized numerical attributes described with crisp, interval and fuzzy values and furthermore it allows the existence of missing values in all of them. Algorithm 1 describes the whole process.

In the step 1, all examples in the root node have an initial weight equal to 1, less the examples with set-valued class whose weight will be initialize as we indicate in the Section 2.1. The tail is a priority tail, ordered from higher to lower according to the total weight of the examples of nodes that form the tail. Thus the domain is guaranteed to partition according to the most relevant attributes.

In the step 3, when we expand a node according to an attribute:

1. If the attribute is already discretized, the node is expanded into many children as possible values the selected attribute has. In this case, the tree's behaviour is similar to that described in the Subsection 2.1.
2. If the attribute is not previously discretized, its possible descendants are obtained. To do this, as in C4.5, the examples are ordered according to the value of the attribute in question. To carry out the order of data with crisp, fuzzy and interval values, we need an ordering index, [13]. Therefore, we have a representative value for each interval

Algorithm 1. Search of cut points

SearchCrispIntervals($in: E, Fuzzy\ Partition; out: Cut\ points$**)**
begin

(a) Start at the root node, which is placed in the initially empty priority tail. Initially, in the root node is found the set of examples E with an initial weight.
(b) Extract the first node from the priority tail.
(c) Select the best attribute to split this node using information gain expressed in (2) as the criterion. We can find two cases: The first case is where the attribute with the highest information gain is already discretized, either because it is nominal, or else because it had already been discretized earlier by the $Fuzzy\ Partition$. The second case arises when the attribute is numerical and non-discretized. In this case it is necessary to obtain the corresponding cut points.
(d) Having selected the attribute to expand node, all the descendants generated are introduced in the tail.
(e) Go back to step two to continue constructing the tree until there are not nodes in the priority tail or until another stopping condition occurs, such as reaching nodes with a minimum number of examples allowed by the algorithm.

end

and fuzzy value and we can order all values of the non-discretized attributes. The index used is calculated as in (4). Let A_i be a fuzzy set (or interval) in the attribute i of the example e:

$$Y(A_i) = \int_0^1 M(A_{i\alpha}) d\alpha \tag{4}$$

where $Y(A_i)$ is the representative value of the fuzzy or interval data of the attribute i and $M(A_{i\alpha})$ is the mean value of the elements of $A_{i\alpha}$.

This index determines for each fuzzy or interval value a number with which we order all values. Using the crisp and the representative values, we find the possible cut points as a C4.5 tree. The intermediate value between value of the attribute for example e_j and for example e_{j+1} is obtained. The value obtained will be that which provides two descendants for the node and to which the criterion of information gain is applied. This is repeated for each pair of consecutive values of the attribute, searching for the value that yields the greatest information gain. The value that yields the greatest information gain will be the one used to split the node and will be considered as a cut point for the discretization of this attribute. When example e descend to the two descendants, the process carries out is the same that we explain in Section 2.1 and if the value of the attribute is fuzzy or interval, we apply the function (1) to determine the membership of this example e to the descendant nodes, because we only use the representative value of these kind of values to order and to get cut points, but when we need use these values to do some estimates, we use the original value and not the representative value.

2.3 Second Stage: Optimizing Fuzzy Partitions with Imprecise Data

In the second stage of the EOFP Algorithm, we are going to use a genetic algorithm to get the fuzzy sets that make up the partitioning of nondiscretized attributes. We have

decide to use a genetic algorithm, because these algorithms are very powerful and robust, as in most cases they can successfully deal with an infinity of problems from very diverse areas and specifically in Data Mining [5]. These algorithms are normally used in problems without specialized techniques or even in those problems where a technique does exist, but is combined with a genetic algorithm to obtain hybrid algorithms that improve results [7].

The genetic algorithm takes as input the cut points which we have obtained in the first stage, but it is important to mention that the genetic algorithm will decide what cut points are more important to construct the fuzzy partitions, so it is possible that many cut points are not used to obtain the optimal fuzzy partitions. Maximum if the first stage gets F cut points for the attribute i, the genetic algorithm can make up F_i+1 fuzzy partitions for the attribute i. However, if the genetic algorithm considers that the attribute i doesn't have a lot of relevance in the dataset, this attribute won't be partitioned. The different elements which compose this genetic algorithm are as follows:

Encoding. An individual will consist of two arrays v_1 and v_2. The array v_1 has a real coding and its size will be the sum of the number of cut points that the fuzzy tree will have provided for each attribute in the first stage. Each gene in array v_1 represents the quantity to be added to and subtracted from each attribute's split point to form the partition fuzzy. On the other hand, the array v_2 has a binary coding and its size is the same that the array v_1. Each gene in array v_2 indicates whether the corresponding gene or cut point of v_1 is active or not. The array v_2 will change the domain of each gene in array v_1. The domain of each gene in array v_1 is an interval defined by $[0, min(\frac{p_r - p_{r-1}}{2}, \frac{p_{r+1} - p_r}{2})]$ where p_r is the r-th cut point of attribute i represented by this gene except in the first (p_1) and last (p_u) cut point of each attribute whose domains are, respectively: $[0, min(p_1, \frac{p_2 - p_1}{2}]$ and $[0, min(\frac{p_u - p_{u-1}}{2}, 1 - p_u]$.

When $F_i = 2$, the domain of the single cut point is defined by $[0, min(p_1, 1 - p_1]$. The population size will be 100 individuals.

Initialization. First the array v_2 in each individual is randomly initialized, provided that the genes of the array are not all zero value, since all the split points would be deactivated and attributes would not be discretized. Once initialized the array v_2 , the domain of each gene in array v_1 is calculated, considering what points are active and which not. After calculating the domain of each gene of the array v_1, each gene is randomly initialized generating a value within its domain.

Fitness Function. The fitness function of each individual is defined according to the information gain defined in [1]. Algorithm 2 implements the fitness function, where:

- μ_{if} is the membership function corresponding to fuzzy set f of attribute i. Again, we must emphasize that this membership function depends on the kind of attribute. Where if the attribute is numerical or belonging to a known fuzzy partition, the membership function is calculated as we have indicated in 2. On the contrary if the attribute is fuzzy or interval, the membership function is calculated as we show in function (1).
- E_k is the subset of examples of E belonging to class k.

  This fitness function, based on the information gain, indicates how dependent the attributes are with regard to class, i.e., how discriminatory each attribute's partitions

Algorithm 2. Fitness Function

Fitness($in:\ E,\ out: ValueFitness$)
begin

1. For each attribute $i = 1, ..., |A|$:
   1.1 For each set $f = 1, ..., F_i$ of attribute i
   For each class $k = 1, ..., |C|$ calculate the probability $P_{ifk} = \frac{\sum_{e \epsilon E_k} \mu_{if}(e)}{\sum_{e \epsilon E} \mu_{if}(e)}$
   1.2 For each class $k = 1, ..., |C|$ calculate the probability $P_{ik} = \sum_{f=1}^{F_i} P_{ifk}$
   1.3 For each $f = 1, ..., F_i$ calculate the probability $P_{if} = \sum_{k=1}^{|C|} P_{ifk}$
   1.4 For each $f = 1, ..., F_i$ calculate the information gain of attribute i and set $f\ I_{if} = \sum_{k=1}^{|C|} P_{ifk} \cdot \log_2 \frac{P_{ifk}}{P_{ik} \cdot P_{if}}$
   1.5 For each $f = 1, ..., F_i$ calculate the entropy $H_{if} = -\sum_{k=1}^{|C|} P_{ifk} \cdot \log_2 P_{ifk}$
   1.6 Calculate the I and H total of attribute i
   $I_i = \sum_{f=1}^{F_i} I_{if}$ and $H_i = \sum_{f=1}^{F_i} H_{if}$
2. Calculate the fitness as : $ValueFitness = \frac{\sum_{i=1}^{|A|} I_i}{\sum_{i=1}^{|A|} H_i}$

end

are. If the fitness we obtain for each individual is close to zero, it indicates that the attributes are totally independent of the classes, which means that the fuzzy sets obtained do not discriminate classes. On the other hand, as the fitness value moves further away from zero, it indicates that the partitions obtained are more than acceptable and may discriminate classes with good accuracy.

Selection. Individual selection is by means of tournament, taking subsets with size 2.

Crossing. The crossing operator is applied with a probability of 0.3, crossing two individuals through a single point, which may be any one of the positions on the vector. Not all crossings are valid, since one of the restrictions imposed on an individual is that the array v_2 should not has all its genes to zero. When crossing two individuals and this situation occurs, the crossing is invalid, and individuals remain in the population without interbreeding. If instead the crossing is valid, the domain for each gene of array v_1 is updated in individuals generated.

Mutation. Mutation is carried out according to a certain probability at interval $[0.01, 0.1]$, changing the value of one gene to any other in the possible domain. First, the gene of the array v_2 is mutated and then checked that there are still genes with value 1 in v_2. In this case, the gene in v_2 is mutated and, in addition, the domains of this one and its adjacent genes are updated in the vector v_1. Finally, the mutation in this same gene is carried out in the vector v_1.

If when a gene is mutated in v_2 all genes are zero, then the mutation process is not produced.

Stopping. The stopping condition is determined by the number of generations situated at interval $[100, 150]$.

The genetic algorithm should find the best possible solution in order to achieve a more efficient classification.

In the next section we want to show with some computational experiments that it is important construct fuzzy partitions from real data versus transform them because we will lost information and accuracy.

3 Experiments

In this section we are going to show different experiments to evaluate if the fuzzy partitions which are constructed without making any transform of data (EOFP Algorithm) are better than fuzzy partitions which are constructed making certain transformation on imprecise data to convert them in crisp data (OFP_CLASS algorithm). All partitions are evaluated classifying with a Fuzzy Random Forest ensemble (FRF) [3] which is able to handle imperfect data into the learning and the classification phases.

The experiments are designed to measure the behavior of fuzzy partitions used in the FRF ensemble using datasets and results proposed in [11,12] where the authors use a fuzzy rule-based classifier to classify datasets with imprecise data such as missing or interval. Also they use uniform partitions to evaluate the datasets and we are going to show how the results are better when the partitions are fuzzy although they are constructed using the modified dataset instead of the original dataset. Also we are going to show how the results in classification are still better if we don't modify data to construct the fuzzy partitions. Due to we are going to compare with results of [11,12], we define the experimental settings quite similar to those proposed by them.

3.1 Datasets and Parameters for FRF Ensemble

To evaluate fuzzy partitions, we have used real-world datasets about medical diagnosis and high performance athletics [11,12], that we describe in Table 1.

Table 1. Datasets

Dataset	|E|	|M|	I	Dataset	|E|	|M|	I
100ml-4-I	52	4	2	Dyslexic-12	65	12	4
100ml-4-P	52	4	2	Dyslexic-12-01	65	12	3
Long-4	25	4	2	Dyslexic-12-12	65	12	3

Table 1 shows the number of examples ($|E|$), the number of attributes ($|M|$) and the number of classes (I) for each dataset. "Abbr" indicates the abbreviation of the dataset used in the experiments.

All FRF ensembles use a forest size of 100 trees. The number of attributes chosen at random at a given node is $\log_2(|\cdot|+1)$, where $|\cdot|$ is the number of available attributes at that node, and each tree of the FRF ensemble is constructed to the maximum size (node pure or set of available attributes is empty) and without pruning.

3.2 Results

These experiments were conducted to test the accuracy of FRF ensemble when it uses fuzzy partitions constructed from real-world datasets with imperfect values

using EOFP Algorithm. These results are compared with the ones obtained by the GFS classifier proposed in [11], which uses uniform partitions and with the results obtained by FRF ensemble when uses fuzzy partitions constructed with the OPF_CLASS Algorithm.

It is important to clarify that OPF_CLASS Algorithm doesn't work with imperfect data. For this reason, to get the fuzzy partitions of these datasets we have modified the original data. The interval or fuzzy values have been changed by their average values. Therefore we have transformed the interval and fuzzy values in crisp values and of this way the OPF_CLASS Algorithm can work with these datasets.

In these experiments we have used the available datasets in "http://sci2s.ugr.es/ keel/" and the available results in [11,12]. There are datasets from two different real-world problems. The first one is related to the composition of teams in high performance athletics and the second one is a medical diagnosis problem. A more detailed description of these problems may be found in [11,12].

High Performance Athletics. The score of an athletics team is the sum of the individual scores of the athletes in the different events. It is the coach's responsibility to balance the capabilities of the different athletes in order to maximize the score of a team according to the regulations. The variables that define each problem are as follows:

- There are four indicators for the long jump that are used to predict whether an athlete will pass a given threshold: the ratio between the weight and the height, the maximum speed in the 40 meters race, and the tests of central (abdominal) muscles and lower extremities;
- There are also four indicators for the 100 meters race: the ratio between weight and height, the reaction time, the starting or 20 m speed, and the maximum or 40 m speed.

The datasets used in this experiment are the following: "Long-4" (25 examples, 4 attributes, 2 classes, no missing values and all attributes are interval-valued), "100ml-4-I" and "100ml-4-P" (52 examples, 4 attributes, 2 classes, no missing values and all attributes are interval-valued).

As in [11], we have used a 10 fold cross-validation design for all datasets. Table 2 shows the results obtained in [11] and the ones obtained by the FRF ensemble with the six combination methods which are explained in detail in [3]. In Appendix 4 we present a brief intuitive description of each of them. Except for the crisp algorithm proposed in [11], in Table 2, the interval $[mean\_min\_error, mean\_max\_error]$ obtained for each dataset according to the decision process described in Section 2.1, is shown. For each dataset, the best results obtained with each algorithm are underlined.

The results obtained in classification by the extended GPS proposed in [11] and FRF ensemble, are very promising because we are representing the information in a more natural and appropriate way, and in this problem, we are allowing the collection of knowledge of the coach by ranges of values and linguistic terms.

The results of FRF ensemble are very competitive with all fuzzy partitions but the fuzzy partitions obtained with EOFP Algorithm are the best.

Table 2. Comparative results for datasets of high performance athletics

		Dataset					
		100ml-4-I		100ml-4-P		Long-4	
	Technique	Train	Test	Train	Test	Train	Test
EOFP Fuzzy partition	FFR_{SM1}	[0.107,0.305]	[0.130,0.323]	[0.043,0.235]	[0.093,0.290]	[0.191,0.484]	[0.083,0.349]
	FRF_{SM2}	[0.110,0.306]	[0.150,0.343]	[0.045,0.237]	[0.110,0.307]	[0.165,0.449]	[0.083,0.349]
	FRF_{MWL1}	[0.070,0.265]	[0.073,0.267]	[0.032,0.224]	[0.060,0.257]	[0.085,0.364]	[0.033,0.299]
	FRF_{MWL2}	[0.060,0.254]	[0.113,0.306]	[0.043,0.235]	[0.060,0.257]	[0.111,0.391]	[0.083,0.349]
	FRF_{MWLT1}	[0.070,0.267]	[0.073,0.267]	[0.032,0.224]	[0.060,0.257]	[0.085,0.364]	[0.033,0.299]
	FRF_{MWLT2}	[0.060,0.252]	[0.093,0.286]	[0.038,0.231]	[0.060,0.257]	[0.107,0.386]	[0,083,0.349]
OFP_CLASS Fuzzy partition	FFR_{SM1}	[0.139,0.331]	[0.150,0.343]	[0.098,0.291]	[0.133,0.310]	[0.120,0.404]	[0.200,0.467]
	FRF_{SM2}	[0.141,0.333]	[0.150,0.343]	[0.096,0.288]	[0.093,0.290]	[0.115,0.391]	[0.200,0.467]
	FRF_{MWL1}	[0.077,0.269]	[0.093,0.287]	[0.075,0.269]	[0.073,0.270]	[0.116,0.396]	[0.100,0.417]
	FRF_{MWL2}	[0.060,0.252]	[0.093,0.287]	[0.077,0.269]	[0.073,0.270]	[0.102,0.382]	[0.100,0.367]
	FRF_{MWLT1}	[0.077,0.269]	[0.093,0.287]	[0.075,0.267]	[0.073,0.270]	[0.107,0.387]	[0.150,0.417]
	FRF_{MWLT2}	[0.062,0.254]	[0.093,0.287]	[0.077,0.269]	[0.073,0.270]	[0.094,0.373]	[0.067,0.333]
	Crisp [11]	0.259	0.384	0.288	0.419	0.327	0.544
	GGFS [11]	[0.089,0.346]	[0.189,0.476]	[0.076,0.320]	[0.170,0.406]	[0.000,0.279]	[0.349,0.616]

Diagnosis of Dyslexic. Dyslexia is a learning disability in people with normal intellectual coefficient, and without further physical or psychological problems explaining such disability. A more detailed description of this problem can found in [11,12].

In these experiments, we have used three different datasets. Their names are "Dyslexic-12', "Dyslexic-12-01" and "Dyslexic-12-12". Each dataset has 65 examples and 12 attributes. The output variable for each dataset is a subset of the labels that follow: - No dyslexic; - Control and revision; - Dyslexic; and - Inattention, hyperactivity or other problems.

These three datasets differ only in their outputs:

- "Dyslexic-12" comprises the four mentioned classes.
- "Dyslexic-12-01" does not make use of the class "control and revision", whose members are included in class "no dyslexic".
- "Dyslexic-12-12" does not make use of the class "control and revision", whose members are included in class "dyslexic".

All experiments are repeated 100 times for bootstrap resamples with replacement of the training set. The test set comprises the "out of the bag" elements.

In Table 3, we show the results obtained when we run FRF ensemble with fuzzy partitions obtained with OFP_CLASS and fuzzy partitions obtained with EOFP for datasets "Dyslexic-12", "Dyslexic-12-01" and"Dyslexic-12-12'.

Also, in Table 3, we compare these results with the best ones obtained in [12] ($(*)$: partition used - four labels; $(**)$ partition used - five labels). Again, in this table, the interval $[mean\_min\_error, mean\_max\_error]$ obtained for each dataset according to

Table 3. Comparative results for datasets of dyslexia

		Dataset					
		Dyslexic-12		Dyslexic-12-01		Dyslexic-12-12	
	Technique	Train	Test	Train	Test	Train	Test
EOFP Fuzzy partition	FRF_{SM1}	[0.000,0.238]	[0.000,0.398]	[0.022,0.223]	[0.039,0.377]	[0.001,0.263]	[0.035,0.422]
	FRF_{SM2}	[0.000,0.228]	[0.000,0.399]	[0.008,0.184]	[0.022,0.332]	[0.009,0.245]	[0.032,0.411]
	FRF_{MWL1}	[0.000,0.270]	[0.000,0.406]	[0.017,0.231]	[0.045,0.383]	[0.001,0.273]	[0.019,0.430]
	FRF_{MWL2}	[0.000,0.270]	[0.000,0.407]	[0.020,0.241]	[0.056,0.385]	[0.001,0.267]	[0.026,0.406]
	FRF_{MWLT1}	[0.000,0.263]	[0.000,0.402]	[0.012,0.216]	[0.038,0.365]	[0.000,0.265]	[0.019,0.427]
	FRF_{MWLT2}	[0.000,0.266]	[0.000,0.404]	[0.015,0.221]	[0.049,0.373]	[0.000,0.262]	[0.024,0.422]
OFP_CLASS Fuzzy partition	FRF_{SM1}	[0.000,0.320]	[0.002,0.511]	[0.000,0.282]	[0.000,0.413]	[0.000,0.405]	[0.000,0.477]
	FRF_{SM2}	[0.000,0.327]	[0.001,0.515]	[0.000,0.253]	[0.000,0.389]	[0.000,0.402]	[0.000,0.469]
	FRF_{MWL1}	[0.000,0.261]	[0.003,0.419]	[0.000,0.264]	[0.000,0.400]	[0.000,0.335]	[0.000,0.422]
	FRF_{MWL2}	[0.000,0.270]	[0.003,0.423]	[0.000,0.276]	[0.000,0.407]	[0.000,0.343]	[0.000,0.414]
	FRF_{MWLT1}	[0.000,0.264]	[0.004,0.419]	[0.000,0.243]	[0.000,0.386]	[0.000,0.331]	[0.000,0.422]
	FRF_{MWLT2}	[0.000,0.267]	[0.003,0.417]	[0.000,0.259]	[0.000,0.394]	[0.000,0.343]	[0.000,0.418]
(*)	Crisp CF^0	0.444	[0.572,0.694]	0.336	[0.452,0.533]	0.390	[0.511,0.664]
(*)	GGFS	–	[0.421,0.558]	–	[0.219,0.759]	–	[0.199,0.757]
(*)	GGFS CF^0	[0.003,0.237]	[0.405,0.548]	[0.005,0.193]	[0.330,0.440]	[0.003,0.243]	[0.325,0.509]
(**)	Crisp CF^0	0.556	[0.614,0.731]	0.460	[0.508,0.605]	0.485	[0.539,0.692]
(**)	GGFS	–	[0.490,0.609]	–	[0.323,0.797]	–	[0.211,0.700]
(**)	GGFS CF^0	[0.038,0.233]	[0.480,0.621]	[0.000,0.187]	[0.394,0.522]	[0.000,0.239]	[0.393,0.591]

the decision process described in Section 2.1, is shown. For each dataset, the best results obtained with each algorithm are underlined.

As comment about all experiments, we see that FRF ensemble with EOFP fuzzy partitions obtains better results in test than FRF with OFP_CLASS fuzzy partitions. FRF ensemble is a significant improvement over the crisp GFS. In these experiments we can see that when the partitions are obtained with the original data using the EOFP algorithm, the accuracy is higher (the intervals of error are closer to 0 and they are less imprecise). As also discussed in [12] is preferable to use an algorithm which is able of learning with low quality data than removing the imperfect information and using a conventional algorithm.

4 Conclusions

In this paper we have presented the EOFP Algorithm for fuzzy discretization of numerical attributes. This algorithm is able to work with imperfect information. We have performed several experiments using imprecise datasets, obtaining better results when working with the original data. Besides, we have presented a fuzzy decision tree which can work with imprecise information.

Our final conclusion, as many papers in the literature are indicating, is that it is necessary to design classification techniques so they can manipulate original data that can be imperfect in some cases. The transformation of these imperfect values to (imputed) crisp values may cause undesirable effects with respect to accuracy of the technique.

Acknowledgements. Partially supported by the project TIN2011-27696-C02-02 of the MINECO of Spain. Thanks to the Funding Program for Research Groups of Excellence (04552/ GERM/06) and the scholarship FPI of Raquel Martínez granted by the "Agencia Regional de Ciencia y Tecnología - Fundación Séneca", Murcia, Spain.

References

1. Au, W.-H., Chan, K.C., Wong, A.: A fuzzy approach to partitioning continuous attributes for classification. IEEE Tran., Knowledge and Data Engineering 18(5), 715–719 (2006)
2. Bonissone, P.P.: Approximate reasoning systems: handling uncertainty and imprecision in information systems. In: Motro, A., Smets, P. (eds.) Uncertainty Management in Information Systems: From Needs to Solutions, pp. 369–395. Kluwer Academic Publishers (1997)
3. Bonissone, P.P., Cadenas, J.M., Garrido, M.C., Díaz-Valladares, R.A.: A fuzzy random forest. Int. J. Approx. Reasoning 51(7), 729–747 (2010)
4. Cadenas, J.M., Garrido, M.C., Martínez, R., Muñoz, E.: OFP_CLASS: An Algorithm to Generate Optimized Fuzzy Partitions to Classification. In: 2nd International Conference on Fuzzy Computation, pp. 5–13 (2010)
5. Cantu-Paz, E., Kamath, C.: On the use of evolutionary algorithms in data mining. In: Abbass, H.A., Sarker, R.A., Newton, C.S. (eds.) Data Mining: A Heuristic Approach, pp. 48–71. Ideal Group Publishing (2001)
6. Casillas, J., Sánchez, L.: Knowledge extraction from data fuzzy for estimating consumer behavior models. In: IEEE Confer. on Fuzzy Systems, pp. 164–170 (2006)
7. Cox, E.: Fuzzy Modeling and Genetic Algorithms for Data Mining and Exploration. Morgan Kaufmann Publishers (2005)
8. Garrido, M.C., Cadenas, J.M., Bonissone, P.P.: A classification and regression technique to handle heterogeneous and imperfect information. Soft Computing 14(11), 1165–1185 (2010)
9. Liu, H., Hussain, F., Tan, C.L., Dash, M.: Discretization: an enabling technique. Journal of Data Mining and Knowledge Discovery 6(4), 393–423 (2002)
10. Otero, A.J., Sánchez, L., Villar, J.R.: Longest path estimation from inherently fuzzy data acquired with GPS using genetic algorithms. In: International Symposium on Evolving Fuzzy Systems, pp. 300–305 (2006)
11. Palacios, A.M., Sánchez, L., Couso, I.: Extending a simple genetic coopertative-competitive learning fuzzy classifier to low quality datasets. Evolutionary Intelligence 2, 73–84 (2009)
12. Palacios, A.M., Sánchez, L., Couso, I.: Diagnosis of dyslexia with low quality data with genetic fuzzy systems. Int. J. Approx. Reasoning 51, 993–1009 (2010)
13. Wang, X., Kerre, E.E.: Reasonable propierties for the ordering of fuzzy quantities (I-II). Journal of Fuzzy Sets and Systems 118, 375–405 (2001)

Appendix

Combination Methods

We present, with a brief intuitive description, the combination methods used in this paper. These methods are described with more details in [3].

- Method SM1: In this method, each tree of the ensemble assigns a simple vote to the most voted class among the reached leaves by the example. The FRF ensemble classifies the example with the most voted class among the trees.
- Method SM2: The FRF ensemble classifies the example with the most voted class among the reached leaves by the example.
- Method MWL1: This method is similar to SM1 method but the vote of each reached leaf is weighted by the weight of the leaf.
- Method MWL2: In this case, each leaves reached assigns a weight vote to the majority class. The ensemble decides the most voted class.
- Method MWLT1: This method is similar to MWL1 method but the vote of each tree is weighted by a weight assigned to each tree.
- Method MWLT2: Each leaf reached vote to the majority class with a weighted vote with the weight of the leaf and the tree to which it belongs.