Bottom-Up Fuzzy Partitioning in Fuzzy Decision Trees

Maciej Fajfer
Dept. of Mathematics and Computer Science
University of Missouri – St. Louis
St. Louis, Missouri 63121
maciejf@kme.pl

Cezary Z. Janikow
Dept. of Mathematics and Computer Science
University of Missouri – St. Louis
St. Louis, Missouri 63121
janikow@umsl.edu

Abstract

FID is a publicly available fuzzy decision tree software for classifying fuzzy data. This paper describes a new domain partitioning technique, bottom-up, which has just been implement to complement the previously available top-down technique.

1. Introduction

Decision trees are one of the most popular methods for learning and reasoning from feature-based examples [7]. However, they often have been criticized for their persistent over-reliance on near-perfect data, and for the resulting degradation in the presence of imperfect data. Data imperfection might have been the result of noise, imprecise measurements, subjective evaluations, inadequate descriptive language, or missing data. Additional problems arise from continuous or simply large nominal attributes - all such domains have to be partitioned. Some of these potential problems have been successfully addressed in the past. For example, Quinlan has proposed some methods for dealing missing features both in training data and in the examples to be classified [8]. Continuous domains have been addressed by CART [1] and subsequently by C4.5, along with tree pruning techniques [9].

A more recent method is to combine fuzzy representation, and in particular its ability to provide comprehensible descriptive language, and its approximate reasoning techniques, with decision trees. The result is a fuzzy decision tree, such as that described in [3].

The methodology consists of three elements. First, there are two domain partitioning methods: one is top-down, creating local and minimal partitioning needed for generating a fuzzy tree, the other is bottom-up, creating more global partitioning. Here, domain partitioning is performed prior actual tree generation. Second, there is a procedure for building fuzzy decision tree, which tree can also be interpreted as a set of fuzzy rules. The tree can be built using a number of potential fuzzy norms. Finally, there are a number of inference rules, for assigning classifications to new samples - based on the information extrapolated from the tree. The inferences fall into two basic categories: set-based (following local inferences in fuzzy rules) and exemplar-based (following exemplar-based learning). The software can handle a mixture of features: symbolic, fuzzy terms, and numeric, and it can reason under incomplete/missing information.

This paper describes the new bottom-up domain partitioning technique. The previously available technique was aimed at producing the minimal set of partitions locally necessary for minimizing tree size [4]. As such, it was a top-down technique, implementing domain splitting rules while building a tree. However, the fuzzy decision tree has just been extended to a decision forest (described separately), which uses redundant knowledge for performing more elaborate classification. This redundancy requires more global rather than minimal local partitioning. Such a domain partitioning is presented here.

2. Fuzzy Decision Trees

Decision tree methods use recursive partitioning procedures to build decision trees. Subsequently, they use matching inference procedures for classification of new samples.

ID3 [7], and its successor C4.5 [9], along with CART [1], are the two most widely used decision trees. Their basic ideas are the same: partition the sample space in a data-driven manner, and represent the partition as a tree. An important property of these algorithms is that they implicitly attempt to minimize the size of the tree while optimizing some local quality measure - such as entropy or gini index [1][7][9].

The tree is constructed in a data-driven algorithm. Each node in the tree represents a subspace of the event space, and thus the whole tree is a partitioning procedure on the event space. In each node, the partitioning continues by selecting the best decision (an attribute or a relation) to further partition the subspace. When a single attribute is selected for the split decision, the most commonly used

selection method is information gain, which is computationally simple and effective: select an attribute for testing (or a new threshold on a continuous domain) such that the information difference between that contained in a given node and in its children nodes (resulting from splitting according to those tests) is maximized. The information contents is measured according to [7]: $I^N = -\sum_{i=1}^{|C|}(p_i \cdot \log p_i)$, where C is the set of decisions, and p_i is the probability that a training sample in the node represents class i.

1. The root of the decision tree contains all training examples. It represents the whole description space since no restrictions are imposed yet.
2. Work with any node N. The node becomes a leaf when either its samples come from a unique class, when all attributes are used on the path leading to the node, or when possibly information in the node becomes too unreliable (e.g., when too few examples are found). Proceed only when decided to further split the node.
3. Compute the information content at the node N. Then, for each attribute a_i not appearing on the path to N and for each of its domain values a_{ij}, compute the information contents in children nodes restricted by the additional condition $a_i=a_{ij}$. Subsequently, compute a combined weighted information contents in all the children, and the resulting gain with respect to N. Note that for the not-pre-partitioned domains, the algorithm tries all applicable thresholds, selecting the best one - this results in a binary test (note that such attributes can appear more than once on a given path, with different binary test).
4. Select the attribute maximizing the gain, and subsequently split node N using the applicable tests.

Afterwards, decision trees use the same basic inference mechanism for classifying new samples. Features of a sample are matched against the tests of the tree, starting from the root and descanting along the matching path. The sample is classified according to the classification of the leaf that it reaches.

Fuzzy decision trees differ in two respects: they use splitting criteria based on fuzzy restrictions, and their inference procedures are different. Fuzzy sets defining the fuzzy terms used for building a tree are imposed on the algorithm (or generated in the domain partitioning stage).

FID is a widely used publicly available fuzzy decision tree. It can handle data described by various kinds of attribute domains. For example, the same data can be described by a mixture of nominal attributes and attributes with continuous domains - which can be prepartitioned by the user or not. The same mixture of values can be used as classes values. Each piece of data is also augmented with a "confidence" weight (disregarded here in subsequent presentation).

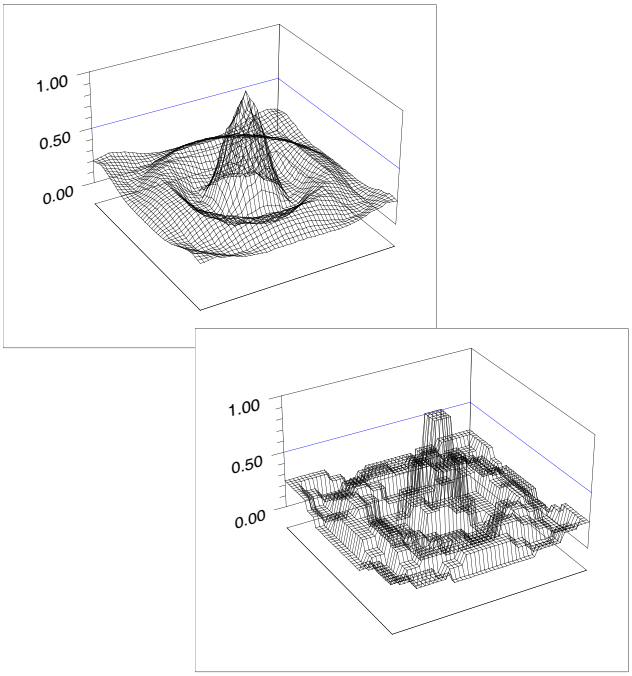
If some domains lack partitioning, the algorithm performs the preprocessing, in which either the top-down or the bottom-up technique is applied to partition such domains. After the preprocessing, the fuzzy decision tree is constructed similarly to the standard decision tree, with a recursive depth-first procedure. However, there are a number of subtle differences:

1. Data samples may match more than one test of a node. When aggregated over multiple levels, this leads to samples falling into many nodes, with a real-valued degree (based on the aggregated match to the fuzzy restrictions on the path).
2. The information contents formula is modified to reflect partial memberships (in addition to allowing absent features). For details, see [3].
3. Fuzzy match is determined based on preselected norms, or by selecting best norms from a predefined set.

However, the most profound differences are in the process of classifying a new sample. These differences arise from the fact that

- FID trees have leaves that are more likely to contain samples of different classes (with different degrees of match),
- the inference procedure is likely to match the new sample against multiple leaves, with varying degrees of match.

Figure 1 Illustrations of the knowledge of two FID trees trained for the mexican sombrero function.



To account for these potential problems, a number of inferences routines have been proposed. Some inferences follow the ideas of approximate reasoning [3], other follow machine learning principles of exemplar learning [4]. Some of these inferences have more global character, some are more local and behave like noise filters [3]. Whatever specific inference is used, the outcome is a value from the domain of the class variable.

FID trees have been shown to be capable of producing knowledge which is both comprehensible yet capable of generating finer levels of detail - depending of the actually used inferences. For more information, see [3].

As an illustration of the descriptive power of FID trees, consider the well known mexican sombrero function [10]. When the FID tree is trained with data samples from a 13x13 grid, using domains with predefined 13 fuzzy terms, two of its interpretations, following two different inferences, are illustrated in Figure 1.

3. Fuzzy Partitioning

If at least one attribute does not have a predefined fuzzy partitioning (does not have the linguistic domain), data-driven preprocessing is invoked in order to partition such domains for relevant attributes (and not necessarily all such attributes).

All attributes with predefined partitions retain their given partitions. The remaining attributes are partitioned with fuzzy sets. There are two different methods available. One, reported previously [4], only partitions attributes relevant to building a decision tree. Below we describe the new method, which attempts to partition all domains in a more global way.

4. Bottom-up Partitioning

The bottom-up partitioning is a global data-driven partitioning strategy. It is aimed at partitioning all continuous domains. Moreover, based on some user parameters, each domain may be forced to have a number of partitions in some predefined range (between `minNumLingVals` and `maxNumLingVals`).

The process starts with a maximal partitioning (at the data level, that is each attribute-value is treated as an individual partition of the given domain) and then uses heuristics to generalize that partitioning. The algorithm consists of three main stages: clustering, safe merging and merging. In the initial clustering stage, every data event is assigned to an individual cluster. Then iteratively, two nearest clusters, using some distance measure, are joined as long as a given error measure is not exceeded.

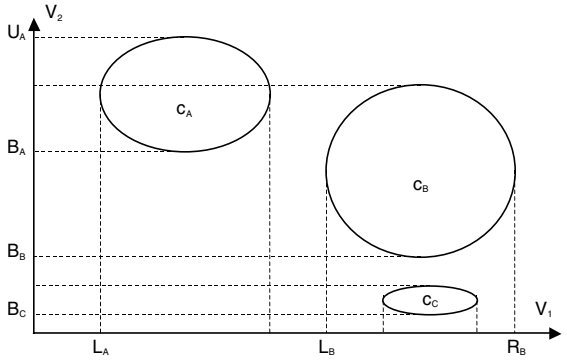
The clustering stage ends with projecting the clusters onto the domains. This creates initial partitioning for the merging stage. Safe merging combines neighboring partitions only if they contain data of exactly the same classification. It is followed with the proper merging, in which entropy measure and error rate are both used to select partitions to be joined. The process stops when no more joins can be made under the allowed error rate.

This algorithm is an extension of a similar algorithm proposed by Grzymala-Busse [2].

4.1 Clustering

The set of training examples is $E = \{e_j | e_j = (u_j^1, \ldots u_j^n, y_j)\}$, where y_j is the fuzzy class or continuous decision value, $j = 1, \ldots N$, where N is number of examples and n is the number of attributes used in discretization (only those subject to discretization). The set of clusters is $C = \{c_D\}$, $D = 1, \ldots m$, where m is the number of clusters.

Figure 2 Illustration of projection of clusters.



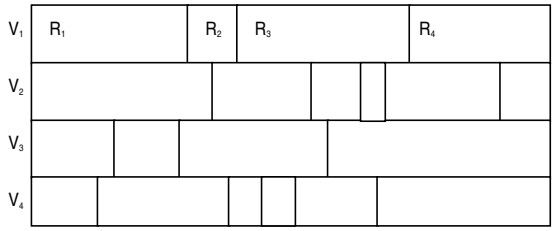
1. We start with N clusters. Each cluster consists of one example and it is a point in the hyperspace defined by the set of attributes.
2. Compute the distance between clusters i and j as: $d_{ij} = \sqrt{\sum_{k=1}^{n} (u_i^k - u_j^k)^2}$
3. Join two closest clusters c_A, c_B into c_{AB}, and compute new distances to all remaining clusters using the following formula
$$\forall (c_D \in C, c_D \neq c_{AB}) d_{AB,D} = \frac{d_{AD} + d_{BD}}{2} + \frac{d_{AB}}{4}$$
4. For each cluster, compute the error rate
$$E_{c_D} = \frac{\sum_{k=1}^{|D_C|} P_k^{c_D} - max_{k=1,\ldots|D_c|}(P_k^{c_D})}{\sum_{k=1}^{|D_C|} P_k^{c_D}}$$

5. If the global error rate $E = \sum_{c_D \in C} \frac{\sum_{k=1}^{|D_C|} P_k^{c_D}}{\sum_{k=1}^{|D_C|} P_k^{C}} E_{c_D}$ is greater than a parameter `ClusterStop`, stop clustering, undo the last join and go to the next step, otherwise go back to step 3.
6. Project the clusters onto domains and create partitions. Figure 2 illustrates partitions for attribute V_1: $[L_A, L_B)[L_B, R_B]$, and V_2: $[B_C, B_B)[B_A, U_A]$, generated by clusters c_A, c_B. To define partitions we use the minimal boundary of every cluster on a given attribute and the maximal boundary of all maximal boundaries (skipping clusters which define subdomains of any other cluster).

4.2 Safe Merging

Safe merging is the process of joining neighborhood partitions under the condition that they contain examples belonging to exactly the same class. For example, partitions R_1 and R_2 in Figure 3 would be merged if they contained examples of exactly the same class. We repeat the process for all possible unions.

Figure 3 Safe Merging.



4.3 Merging

Merging is the final joining process, where partitions containing different classes can be joined - the process is driven by heuristics (entropy) and error measures.

1. Compute entropy in every partition R_i: $I^{R_i} = -\sum_{k=1}^{|D_C|} \left(\frac{P_k^{R_i}}{P^{R_i}} \log \frac{P_k^{R_i}}{P^{R_i}} \right)$.
2. For all possible unions compute the resulting entropy as $I^{R_i, R_j} = \frac{P^{R_i}}{(P^{R_i} + P^{R_j})} I^{R_i} + \frac{P^{R_j}}{(P^{R_i} + P^{R_j})} I^{R_j}$.
   However, union on a given attribute is possible if the number of partitions is still greater than the parameter `minNumnLingVals`.
3. Find the union with minimum I^{R_i, R_j} over all attributes.
4. Join partitions *i* and *j*. Compute error rate for all hyper subspaces defined by partitions on all attributes $E_{hs_i} = \frac{\sum_{k=1}^{|D_C|} P_k^{hs_i} - max_{k=1,\ldots|D_C|}(P_k^{hs_i})}{\sum_{k=1}^{|D_C|} P_k^{hs_i}}$
   and global error rate $E = \sum_{hs_i \in HS} \frac{\sum_{k=1}^{|D_C|} P_k^{hs_i}}{\sum_{k=1}^{|D_C|} P_k^{HS}} E_{hs_i}$
5. If E is less than parameter `MergeStop` then accept the union, update needed entropies and go to step 3. Otherwise, undo the union, select next union with minimum I^{R_i, R_j}, and go to step 4. If no more unions can be selected, go to step 6.
6. If there are attributes on which the number of partitions is greater than parameter `maxNumLingVals` then continue merging these attributes as long as needed, regardless of the error rate.

4.4 Creation of Fuzzy Terms on Partitions

Let us assume that we have *n* partitions $R_1, \ldots R_n$ on a given attribute, where $R_i = (a_i, b_i)$. Partition R_i generates a fuzzy set, at present trapezoidal, in such as way the fuzzy set intersects neighboring sets (for R_{i-1} and R_{i+1}) at a_i and b_i, respectively (except for the first and the last sets).

5. Summary

We have described a method to partition continuous or large-valued domains into fuzzy sets. The method is data driven and each domain is partitioned in a data-driven manner. The method is bottom-up as it starts with maximal partitions, which are subsequently refined and merged. The method is especially applicable to just released upgrade to the fuzzy decision forest, which methodology relies on redundant knowledge and thus requires partitioning of all attributes. Therefore, the resulting partitioning are evaluated in the context of the decision forest, reported separately.

A public release of `FID4.0` is available at `http://www.cs.umsl.edu/~janikow/fid/`.

References

[1] L. Breiman, J.H. Friedman, R.A. Olsen & C.J. Stone. *Classification and Regression Trees*. Wadsworth, 1984.

[2] M.R. Chmielewski and J.W. Grzymala-Busse. Global Discretization of Continuous Attrributes as Preprocessing for Machine Learning. In T.Y. Lin and A.M. Wilderberger, (eds), *Soft Computing: Rough Sets, Fuzzy Logic, neural Networks, Uncertainty Management, Knowledge Discovery*, 1995, pp. 294-297.

[3] C.Z. Janikow. Fuzzy Decision Trees: Issues and Methods, *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 28, Issue 1, pp. 1-14, 1998.

[4] C.Z. Janikow and M. Faifer. Fuzzy Partitioning with FID3.1. *Proceedings of the 18th International Conference of the North American Fuzzy Information Society, IEEE* 1999, pp. 467-471.

[5] C.J. Merz, P.M. Murphy. Repository of machine learning databases. Univ. of CA, Dept. of Information and Computer Science, 1996.

[6] R.S. Michalski. Understanding the Nature of Learning. In *Machine Learning: An Artificial Intelligence Approach*, R. Michalski, J. Carbonell & T. Mitchell (eds.), Vol, II, pp. 3-26. Morgan Kaufmann, 1986.

[7] J.R. Quinlan. Induction on Decision Trees. *Machine Learning*, Vol. 1, 1986, pp. 81-106.

[8] J.R. Quinlan. Unknown Attribute-Values in Induction. In *Proceedings of the Sixth International Workshop on Machine Learning*, 1989, pp. 164-168.

[9] J.R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA. 1993.

[10] I. Suh, Hong & T.W. Kim. Fuzzy Membership Function Based Neural Networks with Applications to the Visual Servoing of Robot Manipulators. I*EEE Transactions on Fuzzy Systems*, Vol. 2, No. 3, 8/1994, pp. 203-220.