Automatically Determine Initial Fuzzy Partitions for Neuro-Fuzzy Classifiers

Frank Klawonn and Detlef D Nauck

***Abstract*— Learning a fuzzy classifier from data is a well-known technique in fuzzy data analysis and many learning algorithms have been proposed, typically in the area of neuro-fuzzy systems. All learning algorithms require a number of parameters to be set by the user. These are typically initial fuzzy partitions for all variables and sometimes also the number of fuzzy rules. Especially, for neuro-fuzzy algorithms the initial choice of parameters can be crucial and if ill-chosen may lead to failure of the learning algorithm. Recent trends in data analysis show that automation is an important issue because it helps to provide advanced analytics to users who are no data analysis experts. In order to fully automate a learning algorithm for fuzzy classifiers we preferably need an algorithm that can determine a suitable initial fuzzy partition for the learning algorithm to start with. In this paper we propose such an algorithm that we have implemented to extend the neuro-fuzzy approach NEFCLASS. NEFCLASS has recently been integrated into an automatic soft computing platform for intelligent data analysis (SPIDA).**

I. INTRODUCTION

The design of a fuzzy system requires the definition and choice of a variety of parameters. When constructing a fuzzy system from data, the user is usually required to specify the number of fuzzy sets and their initial shape for each variable. Without detailed knowledge of the data, this leads to a tedious trial and error strategy in finding the appropriate (number of) fuzzy sets.

When the notion of fuzzy sets was introduced by L.A. Zadeh [[16]], his original intention aimed at modeling human expertise incorporating vague knowledge. In recent years, another aspect of fuzzy system has become more and more important. Instead of modeling human experts, fuzzy systems are used to extract knowledge from data. Strong emphasis is put on the interpretability of a fuzzy system learned from data, even for the sake of a certain loss of accuracy in approximating the data.

This paper looks at the problem of determining suitable initial fuzzy sets for fuzzy classifiers that are created from data by a learning process.

When a fuzzy system should be automatically constructed from data, a number of parameters have to be fixed.

Frank Klawonn is with Applied University of Braunschweig / Wolfenbuettel, Department of Computer Science, Salzdahlumer Str. 46/48, D-38302 Wolfenbuettel, Germany (e-mail f.klawonn@fh-wolfenbuettel.de).

Detlef D Nauck (corresponding author) is with BT Group, Chief Technology Office, Research and Venturing, Intelligent Systems Research Centre, Adastral Park, Orion Building pp1/12, Ipswich IP5 3RE, UK (e-mail detlef.nauck@bt.com).

We consider classification tasks of the following form. We have a data set of n data

$$\{\mathbf{x}_1,\ldots,\mathbf{x}_n\} \subseteq \prod_{j=1}^{p}\left(I_j \cup \{?\}\right).$$

Each datum $\mathbf{x}_i$ has p real-valued attributes lying in the intervals $I_1, \ldots, I_p$, but we also allow for missing values in one or more attributes indicated by the symbol '?'. Obviously, any integer-valued attribute or even a categorical attribute can be encoded in terms of a real-valued attribute.

A class is assigned to each datum. We assume that we have c classes that are numbered $\{1,\ldots,c\}$. $C(\mathbf{x}_i)$ denotes the class assigned to $\mathbf{x}_i$. A classifier is a mapping

$$K : \prod_{j=1}^{p}\left(I_j \cup \{?\}\right) \to \{1,\ldots,c\}.$$

If a sufficient number of data is available, a classifier will be trained on the basis of one part of the data set and it is then evaluated with respect to the misclassifications counted on the data not used for learning.

For a fuzzy classifier we have to specify suitable fuzzy sets $\mu_1^{(j)},\ldots,\mu_{m_j}^{(j)}$ on each interval I_j and a set of rules of the form "If attribute j_1 is $\mu_1^{(j)}$ and … and attribute j_r is $\mu_{j_r}^{(j)}$ then class is k", where $k \in \{1,\ldots,c\}$ is the number of the corresponding class and the $\mu_i^{(j)}$ are fuzzy sets defined on the ranges of the corresponding attribute. Note that we do not require that all attributes occur in a rule. It is sufficient that the rule premise refers to a subset of the attributes. Of course, in applications the fuzzy sets in the rules will be labeled by suitable linguistic terms like, for example, small, medium, large, etc.

Given a datum $\mathbf{x} \in \prod_{j=1}^{p}\left(I_j \cup \{?\}\right)$ a single rule is evaluated by computing the minimum of the membership degrees of all (in the rule mentioned) attribute values. If $\mathbf{x}$ has a missing value, the membership degree to the corresponding fuzzy set is assumed to be one [1].

For each class we determine a membership degree of $\mathbf{x}$ by the maximum value of all rules that point to the corresponding class. The fuzzy classifier assigns $\mathbf{x}$ to the class with the highest membership degree.

The evaluation of the rules in terms of a max-min inference scheme could also be replaced by any other suitable combination of a t-conorm and a t-norm.

In order to specify a fuzzy classifier, we have to determine

- the number of fuzzy sets for each attribute,
- the shape of the fuzzy sets,
- the number of rules, we want to use and
- the structure of each rule.

Learning fuzzy classification rules from data can be done, for example, with neuro-fuzzy systems like NEFCLASS [10]. In order to derive a classifier the neuro-fuzzy system requires the specification of the number of fuzzy sets for each attribute and initial fuzzy sets. This is a critical design factor and typically the user is responsible for this task. After this step, based on these fuzzy sets, a rule base can be learned and the fuzzy sets are then optimized. Finally, pruning of rules and fuzzy sets is carried out.

Although certain redundancies can be eliminated in the pruning step, a bad choice of the initial fuzzy sets can slow down the learning process significantly or even let the training algorithm get stuck in a local minimum. By providing an algorithm that generates suitable initial fuzzy sets automatically from data we hope to address to issues.

1. Neuro-fuzzy learning will hopefully improve and become faster.
2. The learning algorithm to create a fuzzy classifier from data becomes fully automatic and requires no user intervention at all.

The algorithms for creating fuzzy partitions are based on prior work by Fayyad & Irani [5] (computation of boundary points for non-fuzzy intervals) and Elomaa & Rouso [2], who provided algorithms for computing optimal (non-fuzzy) interval partitions if the problem is characterized by a small low-dimensional data set (for subsequent improvements see [3] and [4]). Applications of these algorithms are know from the area of fuzzy decision trees [14],[17].

The new aspects described in this paper are

- the creation of fuzzy partitions based on interval partitions
- a new heuristics to compute nearly optimal partitions for large data sets and/or many boundary points
- a method to reduce fuzzy partitions by considering combinations of attributes
- a method to reduce fuzzy partitions for high-dimensional problems by considering pairs of attributes.

II. Discretisation and Fuzzy Partitions

Before we can create a fuzzy classifier by using a neuro-fuzzy learning procedure, we must specify fuzzy partitions, i.e. the number, shape and position of fuzzy sets, for each attribute of a transaction. In the following we describe in detail how this can be done automatically.

When we consider a fuzzy classifier that uses only a single attribute then the classification will partition the range of the attribute into disjoint intervals. This is at least true, if the fuzzy sets satisfy typical restrictions, for instance that they are unimodal and that never more than two fuzzy sets overlap. A typical choice of fuzzy sets is depicted in Figure 1.

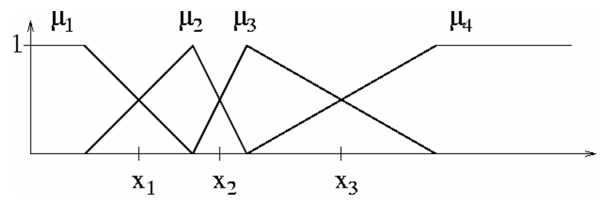


Figure 1: A typical fuzzy partition

In this case, fuzzy set μ_1 prevails for values less than x_1, μ_2 for values between x_1 and x_2, μ_3 for values between x_2 and x_3, and μ_4 for values larger than x_3.

The situation is different, if more than one attribute is considered. A fuzzy partition as shown in Figure 1 induces a partition into disjoint intervals for one attribute. From these interval partitions we obtain a partition of the product space of all attribute ranges into hyper-boxes. If all possible rules are used and each rule is referring to all attributes, the resulting classifier will assign a class to each hyper-box [7]. If not all rules are used, class boundaries can be found within hyper-boxes.

A. Finding a Partition for a Fixed Number of Intervals

Having in mind the view of a classifier based approximately on a partition of the input space into hyper-boxes, we can see an analogy to decision trees. Standard decision trees are designed to build a classifier using binary attributes or, more generally, using categorical attributes with a finite number of values. In order to construct a decision tree in the presence of real-valued attributes, a discretisation of the corresponding ranges is required. The decision tree will then perform the classification task by assigning classes to the hyper-boxes (or unions of these hyper-boxes) induced by the discretisation of the attributes.

The task of discretisation for decision trees is guided by the same principle as the construction of the decision tree itself. In each step of the construction of the decision tree the attribute is chosen for a further split that maximizes the information gain which is usually defined as the expected reduction in entropy.

Generalizing a method for binary splits by Fayyad and Irani [5], Elomaa and Rousu [2] proposed a technique for splitting/discretisation of a range into more than two intervals.

The problem can be defined as follows, when data with a missing value in the considered attribute are simply ignored. We consider a single attribute j and want to partition the range into a fixed number t of intervals. This means that we have to specify t-1 cut points T_1, …, T_{t-1} within the range. The cut points should be chosen in such a way that the entropy of the partition is minimized. Let T_0 and T_t denote the left and right boundary of the range, respectively.

Let us assume that n_i (i =1,…, t) of the n data fall into the interval between T_{i-1} and T_i, when we consider only the jth attribute. Let k_q denote the number of the n_i data that belong to class q. Then the entropy in this interval is given by

$$E_i = -\sum_{q=1}^{c} \frac{k_q}{n_i} \cdot \log\left(\frac{k_q}{n_i}\right) \qquad (1)$$

The overall entropy of the partition induced by the cut points is then the weighted sum of the single entropies

$$E = \sum_{i=1}^{t} \frac{n_i}{n} \cdot E_i \qquad (2)$$

which should be minimized by the choice of the cut points. n is the number of data where attribute j does not have a missing value.

If we sort the data with respect to the values in the jth attribute, it was proved in [2] that for an optimal splitting we only have to consider boundary points as cut points.

We call a value T in the range of attribute j a boundary point, if in the sequence of data sorted by the value of attribute j, there exist two data x and y, having different classes, such that $x_j < T < y_j$, and there is no other datum z such that $x_j < z_j < y_j$.

In the following example (Figure 2) the boundary points are marked by lines.

```
value:  1  2 | 3  3  4 | 5  5 | 6  6 | 7  8  8  9 | 10 | 11  11  12
class:  3  3 | 1  1  1 | 2  2 | 1  3 | 3  3  3  3 |  2 |  1   1   1
```

Figure 2: Boundary Points

Note that it is allowed that different data might have the same values in the considered attribute. Although this situation seldom occurs, when the attribute is really continuous-valued, it is very common for integer-valued attributes.

When we have computed the boundary points, we can construct the optimal discretisation minimizing (2) for a fixed number of intervals. If we have b boundary points and want to split the considered domain into t intervals, we have to evaluate $\binom{b}{t-1}$ partitions. In the worst case, the number of boundary points b equals the number data n-1. But usually we will have $b << n$ so that even in the case of large data sets $\binom{b}{t-1}$ remains a computationally tractable number for small values of t. Nevertheless, if $\binom{b}{t-1}$ is not acceptable in terms of computation time, we apply the following heuristic method to find a partition yielding a small value for (2).

We start with a uniform partition of the range with intervals of the same length or intervals each containing the same number of data. Then we determine, how much each interval contributes to the overall entropy, i.e., referring to equations (1) and (2), we determine for each interval the value

$$-\frac{n_i}{n} \sum_{q=1}^{c} \frac{k_q}{n_i} \cdot \log\left(\frac{k_q}{n_i}\right) = -\frac{1}{n} \sum_{q=1}^{c} k_q \cdot \log\left(\frac{k_q}{n_i}\right) \qquad (3)$$

Based on these values, we enlarge intervals for which (3) is small and we shrink intervals with a high contribution to the entropy. This scaling procedure is repeated until no further improvements could be achieved within a fixed number of steps.

If the number of intervals is fixed, we apply the procedure of Elomaa and Rousu [2] otherwise we switch to the above described scaling heuristics.

B. *Determining the Number of Intervals*

Since we do not want to fix the number of intervals in advance, we have to define a criterion, how many intervals a partition should contain. It is obvious, that the entropy (2) decreases with the number of intervals t, at least when we choose the optimal partition for each t. Therefore, we start with a binary split of two intervals and increase the number of intervals as long as the new partition reduces the entropy compared to the previous partition by a certain percentage or the maximum number of intervals is exceeded.

As long as the method based on the boundary points seems computationally tractable, depending on the number $\binom{b}{t-1}$ mentioned in the previous subsection, we apply the boundary point method otherwise we switch to the above described scaling heuristics.

Figure 7 illustrates the algorithm for this overall strategy that computes suitable partitions for single attributes.

III. From Interval Partitions to Fuzzy Partitions

From the partitions that we have a computed for each attribute, we construct fuzzy sets in the following way. We assume that the partition into t intervals is defined by the cut points T_1, …, T_{t-1}. T_0 and T_t denote the left and right boundary of the corresponding attribute range. Except for the left and right boundaries of each range, we use triangular membership functions, taking their maximum in the center of an interval and reaching the membership degree zero at the centers of the neighboring intervals. At the left and right boundaries of the ranges we place trapezoidal membership functions. They are one between the boundary of the range and the center of the first, respectively, last interval and reach the membership degree zero at the center of the neighboring interval. Figure 3 illustrates the construction of fuzzy sets from interval partitions.

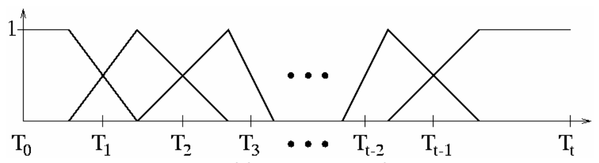


Figure 3: Construction of fuzzy partitions from interval partitions

A. *Partition Simplification*

The construction of the fuzzy sets, respectively the discretisation, was based on the reduction of entropy/information gain, when each variable is considered independently. However, when attributes are correlated, we might further reduce the number of intervals (fuzzy sets). In order to evaluate the information gain of partitions for combinations of variables, we have to consider the partition of the product space into hyper-boxes induced by the interval partitions of the single domains.

In principle, we would have to apply (1) and (2) to hyper-boxes instead of intervals and find the optimal partition into hyper-boxes. In this case, we do not ignore data with missing values, but assign them to larger hyper-boxes corresponding to unions of hyper-boxes. In Figure 4 such a larger box is shown, which is induced by choosing the second (of three) intervals of attribute a_1, the first (of two) intervals of attribute a_2 and a missing value in attribute a_3.

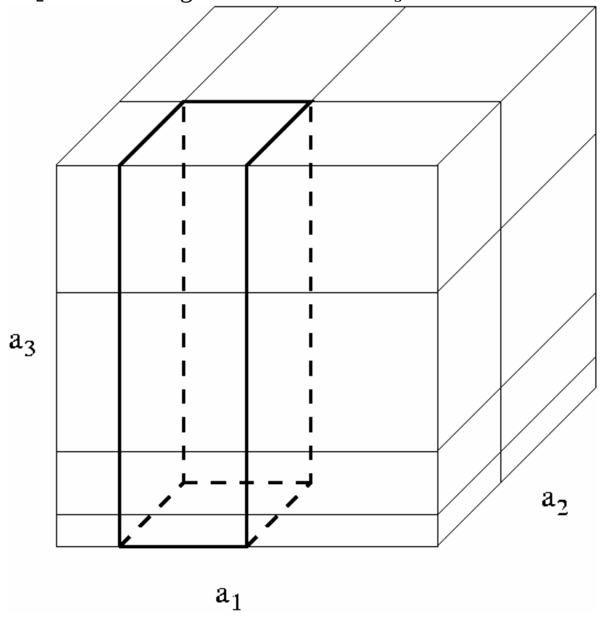


Figure 4: A box induced by a missing value

Unfortunately, the technique of choosing cut points as boundary points does not make sense in multi-dimensional spaces. Our heuristic method of minimizing the overall entropy by scaling the intervals with respect to their entropy, could be applied to the multi-dimensional case as well, but only for the price of an exponential increase of computational costs in terms of the number of attributes. If we have t_j intervals for attribute j (j = 1, …, p), we would have to compute the entropy for $\prod_{j=1}^{p}(t_j+1)$ hyper-boxes for the overall entropy value of one partition into hyper-boxes, including the hyper-boxes representing regions with missing values. In case of six attributes, each one split into three intervals, we would have to consider $(3+1)^6 = 4096$ hyper-boxes for the evaluation of one partition.

Therefore, we do not try to find an overall optimal partition into hyper-boxes, but instead try to simplify the partitions that we have obtained from the single domain partitions. Since the partitions are generated in an incremental way, we do not only store the resulting partitions, but also partitions with fewer intervals. The underlying idea is to check, whether we can go back to a partition with fewer intervals for an attribute without increasing the entropy significantly, when we consider this attribute in connection with other attributes.

First of all, the attributes are sorted with respect to the reduction of entropy that their associated interval partitions provide. For the comparison, required for the sorting, we have to take missing values into account. Let E denote the overall entropy of the data set with n data. Assume that for m_j data attribute j has a missing value. Then the corresponding entropy in terms of (2) would be

$$E = \sum_{i=1}^{t} \frac{n_i}{n-m_j} \cdot E_i$$

if we simply ignore the data with missing values.

In the extreme case that all data except for one have a missing value for attribute j, this entropy would reduce to zero, although the actual information gain by knowing attribute j is almost zero. Therefore, we define

$$\begin{aligned} E &= \frac{n-m_j}{n} \cdot \sum_{i=1}^{t} \frac{n_i}{n-m_j} \cdot E_i + \frac{m_j}{n} \cdot E_{\text{missing}} \\ &= \frac{1}{n} \cdot \sum_{i=1}^{t} n_i \cdot E_i + \frac{m_j}{n} \cdot E_{\text{missing}} \end{aligned}$$

E_{missing} is the entropy of the data with a missing value for the jth attribute. If we assume that missing values occur randomly, E_{missing} will coincide with the overall entropy of the data set.

There are two strategies that we apply, depending on the number of data and the number of hyper-boxes that are induced by the single domain partitions.

The first strategy (Figure 10) is chosen, if the data set is not too large and the number of hyper-boxes is sufficiently small. We start with the attribute whose partition leads to the highest reduction of the entropy and examine the attribute, which was second best in the entropy reduction. We consider the hyper-boxes that are induced by the partition of the ranges of these two attributes. Assume that by our method considering only single attributes, we have found that we should choose t intervals for the attribute that was second best in the entropy reduction. We compare the (hyper-box) entropies in connection with the best attribute, when we use the partition this partition and the partition that we had computed for t –1. If the partition with t –1 intervals does not significantly increase the entropy, we prefer this smaller partition. We even examine the partitions with t –2, t –3 etc intervals, until the increase in entropy seems not acceptable. After that, we examine the attribute that came third in the single domain entropy reduction in connection with the first two attributes, where the second attribute might already have a reduced number of intervals. Then we add the fourth attribute etc.

Since this strategy means that we might have to consider a very large number of hyper-boxes for the last attributes to be investigated, we apply our second strategy (Figure 11), when the first one seems computationally unacceptable. We follow the same principle as in the first strategy, but apply the method only to all pairs of attributes, where in each pair we try to reduce the number of intervals of the attribute with the lesser reduction of entropy.

Finally, Figure 6 shows how to combine the previously introduced algorithms to obtain an overall strategy to compute suitable partitions for all attributes taking their correlations or dependencies into account.

IV. Application in NEFCLASS

We have implemented the described algorithms in the neuro-fuzzy classifier NEFCLASS [8], [9]. NEFCLASS is able to handle missing values, both numeric and symbolic data in the same data set and to determine a rule-base fully automatically. The focus of the NEFCLASS learning algorithms is on creating small interpretable fuzzy rule bases.

The learning algorithm of NEFCLASS has two stages: structure learning and parameter learning. Rule (structure) learning is done by a variation of the approach by Wang and Mendel [15] which was extended to cover also symbolic patterns [9] and to use a rule performance measure for rule selection.

In parameter learning the fuzzy sets are tuned by a backpropagation-like procedure that is based on a simple heuristics instead of a gradient descent approach. After learning NEFCLASS uses pruning strategies to reduce the number of rules as much as possible.

Implementing the algorithms described in this paper was the final missing piece that made NEFCLASS learning fully automatic. The user now is no longer required to provide any initial parameters. Fuzzy partitions and rule base are determined fully automatically. Parameters that control the learning process (learning rates, thresholds, rule learning strategy, pruning strategy) can all be set to suitable values and a user never needs to touch them.

By automating NEFCLASS to that extent it was possible to included it into the automatic intelligent data analysis platform SPIDA which automatically selects, configures and executes data analysis algorithms on behalf of a user [11], [12], [13].

As an example we apply NEFCLASS to the Wisconsin Breast Cancer Data set. If we run NEFCLASS fully automatically, it creates between 1 and 3 fuzzy sets per variable. If only one fuzzy set is created that means the partitioning algorithm has decided not partition that particular variable (mitoses) and the learning algorithm treats it as “don’t care”. The rule learning algorithms selects the best rules per class and determines the number of rules automatically, such that all patterns of the training set are covered by at least one rule. After training and exhaustive pruning is performed (see [8], [9]for a description of the NEFCLASS learning algorithms). The data is randomly separated into training and test sets. The results on the test sets are shown in the following table. NEFCLASS generates the four rule shown below using three variables. There are three fuzzy sets for uni_cell_size (Figure 5) and two fuzzy sets each for bland_chromatin and bare_nuclei.

Table 1: Confusion matrix Breast Cancer Test Data (error = 4.17%)

orig \ pred	**benign**	**malignant**	**Sum**
benign	234	7	241
malignant	8	111	119
Sum	242	118	360

The rules generated are:

Class malignant:

R_0: IF bare_nuclei is large and bland_chromatin is large

R_1: IF uni_cell_size is large and bland_chromatin is large

R_2: IF uni_cell_size is large and bare_nuclei is small and bland_chromatin is large

Class benign:

R_3: IF uni_cell_size is small and bare_nuclei is small and bland_chromatin is small

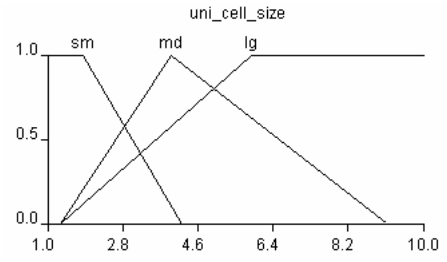


Figure 5: Fuzzy sets for uni_cell_size

If we run NEFCLASS without the automatic partitioning algorithms we have to specify a number of fuzzy sets for each variable. If we choose two fuzzy sets per variable we achieve a test error of 8% with four rules using two variables. If we choose three fuzzy sets we obtain an error of 5.14% with 18 rules using four variables.

V. Conclusions

We have shown how we can automatically determine initial fuzzy partitions in the context of neuro-fuzzy learning.

Fuzzy partitions are created based on interval partitions and we introduced a new heuristics to compute nearly optimal partitions for large data sets and/or many boundary points. In order to achieve small fuzzy classifiers with interpretable rule bases we developed a method to reduce fuzzy partitions by considering combinations of attributes. If this method is too complex to run due to high-dimensional problems we run a simplified version by considering only pairs of attributes.

We have implemented the algorithms into NEFCLASS and shown on a small data set that by providing suitable initial fuzzy partitions we can achieve better results in the subsequent learning process.

REFERENCES

[1] M. Berthold, K.-P. Huber: Tolerating Missing Values in a Fuzzy Environment. In: M. Mares, R. Mesiar, V. Novak, J. Ramik, A. Stupnanova (eds.): Proc. Seventh International Fuzzy Systems Association World Congress IFSA'97, Vol. I, pp 359–362, Academia, Prague (1997).

[2] T. Elomaa, J. Rousu: Finding Optimal Multi-Splits for Numerical Attributes in Decision Tree Learning. Technical Report NC-TR-96-041, Department of Computer Science, Royal Holloway University of London (1996).

[3] T. Elomaa, J. Rousu: General and Efficient Multisplitting of Numerical Attributes. Machine Learning 36, pp 201–244, Kluwer (1999).

[4] T. Elomaa, J. Rousu: Efficient Multisplitting Revisited: Optima-Preserving Elimination of Partition Candidates. Data Mining and Knowledge Discovery 8, pp 97–126, Kluwer (2004).

[5] U.M. Fayyad, K.B. Irani: On the Handling of Continues-Valued Attributes in Decision Tree Generation. Machine Learning 8, pp 87–102 (1992).

[6] Detlef Nauck, Frank Klawonn, and Rudolf Kruse: Foundations of Neuro-Fuzzy Systems. Wiley, Chichester (1997).

[7] L.I. Kuncheva: How Good are Fuzzy If-Then Classifiers? IEEE Transactions on Systems, Man, and Cybernetics, Part B: 30, pp 501–509 (2000).

[8] D. Nauck: Fuzzy data analysis with NEFCLASS. Int. J. Approximate Reasoning 32, pp 103–130, Elsevier (2003).

[9] D. Nauck, Neuro-fuzzy learning with symbolic and numeric data, Soft Computing 8(6) pp 383–396, Springer (2004).

[10] D. Nauck, F. Klawonn and R. Kruse R: ‘Foundations of Neuro-Fuzzy Systems’, Wiley, Chichester (1997).

[11] D. Nauck, M. Spott and B. Azvine, ‘SPIDA – A Novel Data Analysis Tool’, BT Technology Journal 21(4), pp 104–112, Kluwer, Dordrecht (October 2003).

[12] D.D. Nauck, M. Spott, B. Azvine: Fuzzy methods for automated intelligent data analysis. 13th IEEE Int. Conf. Fuzzy Systems (FuzzIEEE 2004), Volume 1, pp. 487–492, Budapest (2004).

[13] M. Spott, D.D. Nauck: On Choosing an Appropriate Data Analysis Algorithm. 14th IEEE Int. Conf. Fuzzy Systems (FuzzIEEE 2005), pp 597– 602, Reno (2005).

[14] Y. Peng, P. Flach. Soft Discretization to Enhance the Continuous Decision Tree Induction. In: C. Giraud-Carrier, N. Lavrac and S. Moyle (eds): Integrating Aspects of Data Mining, Decision Support and Meta-Learning, pp 109–118. ECML/PKDD'01 Workshop Notes, September 2001.

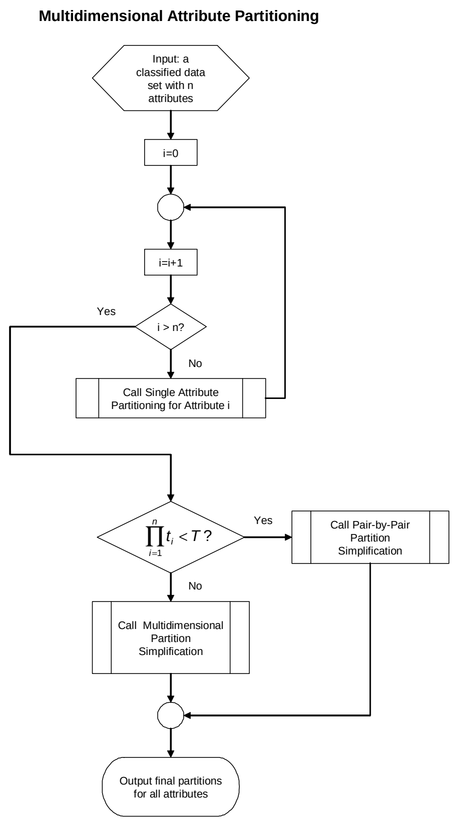
[15] L.-X. Wang, J.M. Mendel, Generating fuzzy rules by learning from examples, IEEE Trans. Syst., Man, Cybern. 22(6) pp. 1414–1427 (1992).

[16] L.A. Zadeh, Fuzzy Sets. Information and Control 8 (1965), 338-353.

[17] J. Zeidler et al.: Fuzzy Decision Trees and Numerical Attributes. Proc. Fifth IEEE Int. Conf. Fuzzy Systems (FuzzIEEE 2005), pp 985–990, IEEE, New Orleans (2005).

APPENDIX (ALGORITHMS)

This section provides flow charts for the partitioning algorithms. For explanations please see the previous sections.



Multidimensional Attribute Partitioning
Input: a classified data set with n attributes
i=0
i=i+1
i > n?
Yes
No
Call Single Attribute Partitioning for Attribute i
$\prod_{i=1}^{n} t_i < T$?
Yes
No
Call Pair-by-Pair Partition Simplification
Call Multidimensional Partition Simplification
Output final partitions for all attributes

Figure 6: Main algorithm for attribute partitioning (*T* is a threshold)

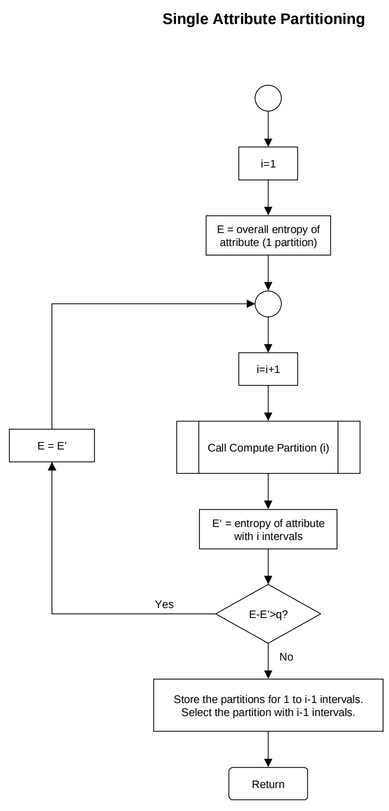

Single Attribute Partitioning
i=1
E = overall entropy of attribute (1 partition)
i=i+1
Call Compute Partition (i)
E' = entropy of attribute with i intervals
E-E'>q?
Yes
E = E'
No
Store the partitions for 1 to i-1 intervals. Select the partition with i-1 intervals.
Return

Figure 7: Algorithm to partition a single attribute (q is a threshold)

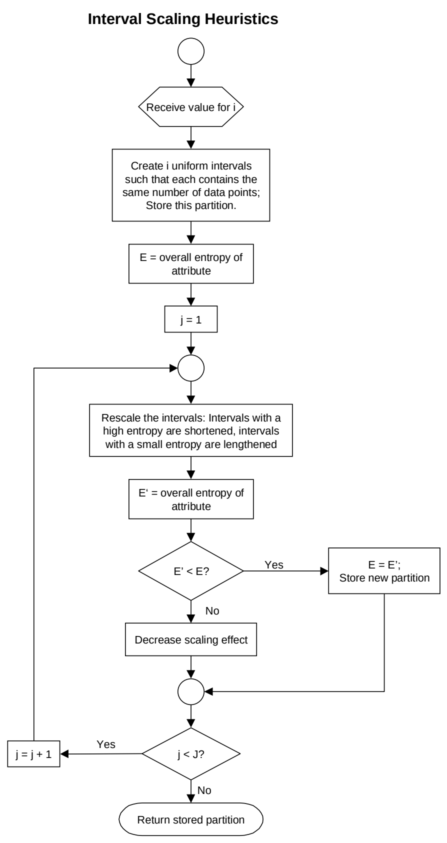

Interval Scaling Heuristics
Receive value for i
Create i uniform intervals such that each contains the same number of data points; Store this partition.
E = overall entropy of attribute
j = 1
Rescale the intervals: Intervals with a high entropy are shortened, intervals with a small entropy are lengthened
E' = overall entropy of attribute
E' < E?
Yes
E = E'; Store new partition
No
Decrease scaling effect
j < J?
Yes
j = j + 1
No
Return stored partition

Figure 9: Heuristics for computing a partition if there are too many boundary points (J is a threshold)

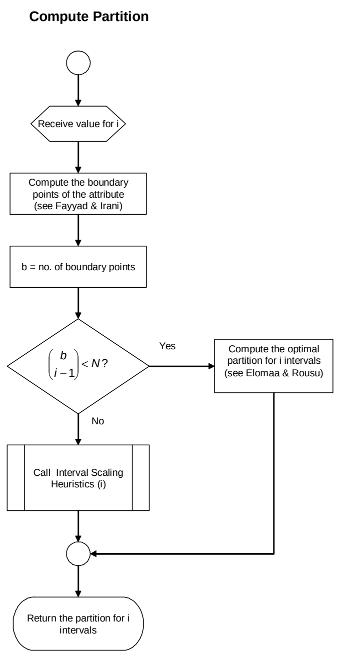

Compute Partition
Receive value for i
Compute the boundary points of the attribute (see Fayyad & Irani)
b = no. of boundary points
$\binom{b}{i-1} < N$?
Yes
Compute the optimal partition for i intervals (see Elomaa & Rousu)
No
Call Interval Scaling Heuristics (i)
Return the partition for i intervals

Figure 8: Compute a Partition (N is a threshold)

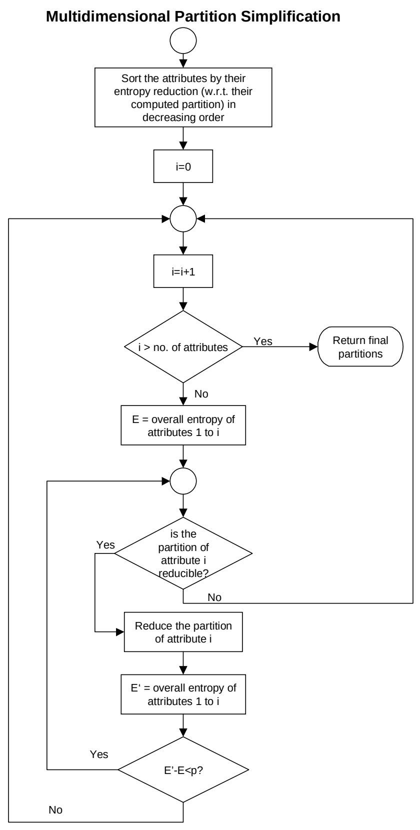

Multidimensional Partition Simplification
Sort the attributes by their entropy reduction (w.r.t. their computed partition) in decreasing order
i=0
i=i+1
i > no. of attributes
Yes
Return final partitions
No
E = overall entropy of attributes 1 to i
is the partition of attribute i reducible?
Yes
No
Reduce the partition of attribute i
E' = overall entropy of attributes 1 to i
E'-E<p?
Yes
No

Figure 10: Algorithm for multidimensional partition simplification (p is a threshold)

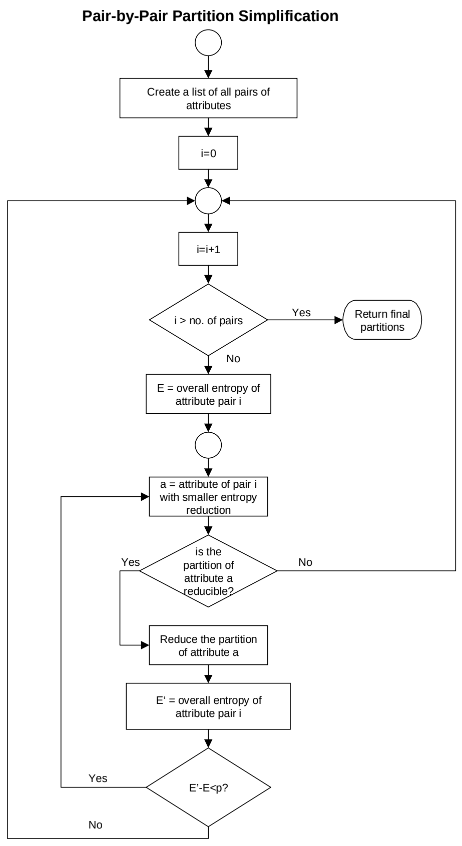

Pair-by-Pair Partition Simplification
Create a list of all pairs of attributes
i=0
i=i+1
i > no. of pairs
Yes
Return final partitions
No
E = overall entropy of attribute pair i
a = attribute of pair i with smaller entropy reduction
is the partition of attribute a reducible?
Yes
No
Reduce the partition of attribute a
E' = overall entropy of attribute pair i
E'-E<p?
Yes
No

Figure 11: Algorithm for pair-by-pair partition simplification (p is a threshold)