

# Knowledge Acquisition from Databases

Xindong Wu  
Monash University, Australia

# Chapter 6

## Dealing With Noise and Real-Valued Attributes

Knowledge acquisition from databases deals with realistic databases, rather than artificial data sets only, so noise and real-valued attributes have to be effectively processed. This chapter analyzes the sources of noise and surveys existing methods to deal with noise and continuous domains. Many of these methods have been implemented in the HCV (Version 2.0) software (see Appendix A).

### 6.1 SOURCES OF NOISE

When designing a learning algorithm, we can normally test it on some artificial data sets, where we can guarantee that the training set for learning is representative so the algorithm can work properly. In these data sets, concepts can normally be defined with first-order logic expressions, which in turn can be used to generate the examples for the training and test sets.

However, a learning algorithm that performs well on artificial data sets does not necessarily work on realistic databases. Noise has become a very important issue in data mining. Before we rush into the design of a noise-tolerant algorithm, we need to have a good understanding of which types of noise we have to face. Noise handling is normally carried out by providing specific facilities to solve the problems caused by these types of noise at different stages of rule induction and interpretation (see Section 6.2). The following is an outline of the sources of noise.

1. Erroneous attribute values. Some data items in the training set were

distorted for one reason or another.

2. Missing attribute values or *Don't Know* values, normally denoted by ?. A *Don't Know* value could take any value in its attribute domain, but we cannot know for sure.
3. Contradictory examples. The same example appears more than once in a data set and is labeled with different classifications at different times. Usually, contradictory examples result from data distortions and misclassifications.
4. Misclassifications. An example was labeled with a wrong classification.
5. Incomplete attributes and uneven distribution of training examples in the example space.
6. Redundant data and *Don't Care* (#) values.

Item 5 is a serious problem with all learning algorithms. When the essential attributes of a problem are not used to describe the training set, a learning system can only try to use other features that are normally unreliable. This is what we call the incomplete attributes problem. The uneven distribution problem occurs when the example space is not well covered by the examples in the training set, or when the examples in the training set are not spread out evenly over the example space, and some regions are not covered at all. We do not normally assume that these two aspects are real problems with our data sets. But once they do happen, the performance of all learning algorithms will generally decrease. In Section 7.4 of Chapter 7, we meet two example sets (primary tumor and Va 5) where none of the algorithms mentioned there have performed well. The best accuracy you can find is 38.2% and 28.2%, respectively.

When there are multiple copies of the same examples in a data set, it is not a problem if all copies have been correctly classified. They become contradictory data if different classifications are attached to the same examples.

Another issue that is often related to noise is *Don't Care* (#) values. # values should not be viewed as noise once they do not contradict with other values or with themselves. However, if an example with *Don't Care* values is converted into a number of equivalent examples that have no *Don't Cares*, and the expanded examples contradict with other examples in the training set, we say that these *Don't Care* values contain noise.

The most serious problem of noise in a data set is that learning algorithms can be misled to produce long and distorted concept descriptions. By trying

to fit every example into the concept descriptions, faulty and noisy examples are included, leading to cluttered and complex concept descriptions. This behavior is commonly known as the *overfitting* problem.

## 6.2 NOISE HANDLING

Noise handling can be carried out at different stages of rule induction and interpretation:

- Preprocessing of training examples before rule induction takes place.
- Induction time processing to avoid overfitting of noisy data.
- Postprocessing of induction results.
- Dealing with unusual examples when interpreting induction results to classify new examples.

The following reviews existing techniques for each of these stages.

### 6.2.1 Preprocessing of Training Data

Preprocessing aims to remove and/or amend some of the noise in the training set, such as contradictory examples and missing information, by specific strategies.

**Contradictions.** Contradictory examples contain significant noise. When trying to produce consistent rules with the training set, these examples should be avoided. In HCV (Version 1.0) [Wu 92e], all contradictory examples are removed during preprocessing in order for them not to confuse the learning algorithm.

However, removing contradictions is not always a good idea, especially in very noisy environments. You might need to get most of the examples removed in some noisy domains if you do not permit any contradictions. Contradictory examples can at least give an indication about which concepts these examples might belong to, so they are useful information for probability analysis.

**Redundant Examples.** As mentioned in Section 6.1, redundant examples are not a problem if they do not form contradictions. Redundancy also provides reliability about the classification of the examples. As mentioned in

Section 4.4 of Chapter 4, redundancy can change the decision trees produced by ID3-like algorithms.

However, sometimes it might be useful to remove redundancy by keeping only unique examples in the training set, to save data space and speed up the induction process.

**Expansion of *Don't Care* Values.** Suppose the  $i$ th attribute has a value domain  $\{v_1, \dots, v_k\}$ . An example  $e$  with a *Don't Care* value of the  $i$ th attribute can be expanded to  $k$  examples, each of which substitutes the *Don't Care* value with a  $v_j$  ( $j = 1, \dots, k$ ).

Expansion of *Don't Care* values is not always useful. Some algorithms like HCV can deal with them very efficiently at induction time. Also, notice that this kind of expansion can bring more contradictory and redundant examples to the training set. If you would like to do the expansion, do it before contradiction and redundancy processing.

**Handling of *Don't Know* Values.** Unknown attribute values can be processed in several ways. The simplest way is to avoid using the corresponding attribute for concept description because the *Don't Know* values could be anything and we do not know for sure.

Another way [Clark & Niblett 89] is to replace the *Don't Know* values with the most frequent attribute value in the training data. Similarly, we can also use the most frequent attribute value for examples of the same class.

The most sophisticated way is to provide facilities for probability analysis of the attribute and assign a set of probabilities rather than a single guessed value to each *Don't Know* value.

**Generation of Nonexistent Examples.** When the number of examples in the training set is not large, and when we would like to produce a conservative description for a concept, we can generate negative examples under the closed world assumption [Lavrac & Dzeroski 94]. Alternatively, if we think the negative examples of the concept are already comprehensive, we can generate positive examples to facilitate induction. This type of generation can reduce the demand of a large number of representative examples in the training set. However, it should be used with great caution, because very few real-world domains are really closed.

### 6.2.2 Induction-Time Processing

The main idea of dealing with noise at induction time is to avoid overfitting of the concept description with the training data.

The number of examples in the training set that are covered by a leaf in a decision tree, or by a conjunctive rule in a rule base, is often called the *weight* of the leaf or the rule [Michalski et al. 86]. A leaf or rule with a very small weight covers few examples in the training set. Conjunctive rules of small weight are called *light complexes* [Michalski et al. 86] or *small disjuncts* [Holte et al. 89]. It has been shown [Ali & Pazzani 92] that small disjuncts often cause an unproportional part of the misclassifications. We can speculate that this is because these light complexes have been created due to noise, and therefore if they are too light we can simply drop them.

**Approximate Partitioning.** Some induction algorithms like HCV partition the positive examples of a concept into a number of groups at first, and then form a description for each group. In these learning algorithms, partitioning is one of the most fundamental steps. In noisy environments, we can allow all the examples in each group to approximately (rather than strictly) to meet some criteria. In the case of HCV, once there is a path that can pass through most rows of the disjunction matrix, we can put all the concerned positive examples into an intersecting group.

**Stopping Criteria.** To avoid overfitting, we can use certain stopping criteria to determine when to stop forming more details of a concept description. When constructing a description for a partition, we do not need to make the description 100% consistent with the positive examples, but instead, cover most of them. In the case of decision tree construction, if most of the examples at one node belong to a specific class, we can stop exploring the node further by simply assigning it to the specific class and treating it as a leaf. This is what we call *prepruning* of decision trees. Prepruning has been used extensively with ID3-like algorithms [Cestnik et al. 87]. A number of criteria including the  $\chi^2$ -test prepruning [Quinlan 93] and the information-based stopping criterion [Quinlan 90] have been designed.

Concept descriptions produced this way do not fit all the examples in the training set. This is what we call *underfitting*. Overfitting and underfitting are two opposite operations in inductive learning. Overemphasizing either of them will cause problems in practice, so we need to find a way to go between. Normally, the user who conducts experiments or runs the learning systems on their specific data sets is expected to specify something like a threshold

value ( $\alpha$ ) to say that they do not require the concept descriptions to be 100% but  $(100 - \alpha)\%$  only consistent with the training examples.

### 6.2.3 Postprocessing of Induction Results

Postprocessing aims to reduce the complexity of induction results and achieve better accuracy. If rule induction has not been facilitated with proper noise handling mechanisms, the results are normally overfitted or affected by the noise in the training set. Postprocessing tries to diminish the effects of the noise by removing parts of the induction results.

Pruning has in most cases proved to be very useful, especially in noisy environments, although one can easily find some cases where pruning is a bad idea. This is consistent with Occam's Razor, which has been widely adopted in machine learning to select between concept descriptions with equal empirical support. For most training sets, there are a large number of competing descriptions (decision trees or rules),<sup>1</sup> all of which equally meet one criterion or another. For example, these descriptions are all consistent with the training set or classify the training set with the same accuracy. To select between them, Occam's Razor suggests that the more compact the better a description is.

This is reasonable in many cases, although one can also easily give adverse examples. Suppose you are asked to distinguish men from women based on the descriptions of a large group of people in terms of gender, height, weight, qualifications, and so on. The right feature for men is *gender = male* and women is *gender = female*. If you cannot catch the right attribute, you normally get more complex and less accurate descriptions such as men have higher salaries and women are not as tall as men.

**Probability Estimation.** Most of the existing work on postprocessing and deduction of induction results is based on probability analysis. Here are some methods mentioned extensively in the literature and adopted in the HCV (Version 2.0) software (see Appendix A) for probability estimation.

- The frequency method. Given that an event has occurred  $n$  times out of  $N$  attempts, the simplest method for estimating the probability of  $e$ ,  $p(e)$ , is to use its relative frequency,  $n/N$ .

---

<sup>1</sup>Some induction algorithms such as CN2 [Clark & Niblett 89] generate a number of concept descriptions at each stage and then choose between them by using some specific criteria such as the naïve bias [Michalski 84], the entropy measure [Clark & Niblett 89], and the significance test or likelihood ratio statistic [Clark & Niblett 89].

- Laplace's Law of Succession. If the data set is representative, the *Laplace's Law of Succession* [Niblett & Bratko 87] uses the following formula rather than the relative frequency to estimate the probability of an event  $e$  under the same assumption as the frequency method:

$$p(e) = \frac{n + 1}{N + 2}. \quad (6.1)$$

- $m$  estimate. The  $m$  estimate method [Lavrac & Dzeroski 94] generalizes the Laplace's formula to the following form:

$$p(e) = \frac{n + m \times p_a(+)}{N + m} \quad (6.2)$$

where  $n$  and  $N$  have the same meanings in the other methods, and  $m$  is a domain dependent variable.  $m$  can be very small if little noise is included in the data set and should grow when the amount of noise becomes substantial.  $p_a(+)$  is a prior probability of  $e$ , which a simpler method such as the Laplace's Law of Succession or the frequency method can be adopted to obtain. When  $m = 0$  the  $m$  estimate becomes the frequency method, and when  $m = 1$  and  $p_a(+)$  is 1 it is equivalent to Laplace's Law of Succession.

**Postpruning of Decision Trees.** Similar to prepruning of decision trees, postpruning can adopt some specific criterion (such as the  $\chi^2$ -test prepruning [Quinlan 93] and the information-based stopping criterion [Quinlan 90]) to check whether a subtree in a decision tree should be replaced by a leaf node.

There are quite a number of postpruning methods in the literature including the cost-complexity pruning, reduced error pruning, and pessimistic pruning described in [Quinlan 87b, Quinlan 93]. Among them, the reduced error pruning seems to be very simple and straightforward. We outline this method in the following.

The reduced error pruning requires a separate pruning set. Given the decision tree  $T$  produced at induction time, we consider every nonleaf subtree  $S$  of  $T$ , and count the number of additional misclassified examples in the pruning set resulting from the subtree  $S$  being replaced by a leaf that takes the classification that the majority of the leaves in  $S$  have taken. If the new tree misclassifies less or an equal number of examples,  $S$  is pruned out. The pruning continues until no further replacements can be carried out without more misclassifications in the pruning set. The pruning set can be taken from the training set.

**Truncation of Rules.** Postprocessing of rules is normally referred to as *truncation*.

TRUNC [Michalski et al. 86] is a very simple rule truncation method, which removes the conjunctions with the least weight (see Section 6.2.2) to avoid overfitting of the data. It does not specify when to stop truncating the rules, and has been used to remove all but one conjunction in every rule. We can easily adapt this strategy for HCV, and use the reduced error criterion. The reduced error pruning method has been used with rules [Brunk & Pazzani 91], operating on selectors and conjunctions instead of subtrees.

**Transformation of Representations.** Another way of performing postprocessing of induction results is to change the representation language. At the same time, measures to reduce the complexity of the concept description already mentioned can be integrated.

Transforming decision trees into production rules has been described in Section 4.5.5 of Chapter 5. Converting variable-valued logic rules to first-order rules has been studied with the AQ17 algorithm [Thrun et al. 91, Wnek & Michalski 94].

## 6.2.4 Deduction-Time Processing

When applying rules produced by HCV or converted from decision trees to a test example, there are three possible cases that demand different actions:

- *No match:* No rules match the example.
- *Single match:* One or more rules indicate the same class match.
- *Multiple match:* More than one rule matches the example and indicates different classes.

The third case does not apply to decision trees produced by ID3-like algorithms, but when the trees are decompiled into production rules [Quinlan 87a, Quinlan 93], the production rules will face the same problem.

In the single match case, the choice of class to the test example is naturally the class indicated by the rules. Deduction-time processing deals mainly with the conflict resolution in the third case and probability estimation for the first case, because the single match case is not a problem at all.

**No match.** In the case of no match, we need to examine the training set and find a class that is close to the test example in question.

- **Largest class.** A common method to deal with no match is to assign all the no match examples to the largest class, called *the default class* [Clark & Niblett 89]. The rationale behind this method is that if the examples in the training set are representative, the possibility of a random example belonging to a large class is higher than that of it belonging to a small one.

The largest class method is good when the number of classes in a training set is small and one of the classes contains a predominant number of examples. However, the results could deteriorate when the number of classes grows, and the number of examples in every class is more evenly spread out.

- **Measure of fit.** Rather than relying solely on the probability of each class in the training set, the *measure of fit* method [Michalski et al. 86] calculates the *MF* value (Measure of Fit) of each class  $c_i$  for each no match example  $e$ .

For a selector  $sel$ ,  $X = [V_1, \dots, V_n]$ , its *MF* value for  $e$  is defined as

$$MF(sel, e) = \begin{cases} 1 & \text{if } sel \text{ is satisfied by } e \\ \frac{n}{|X|} & \text{otherwise} \end{cases} \quad (6.3)$$

where  $n$  is the number of attribute values that the selector includes and  $|X|$  is the number of values in the  $X$  domain.

The *MF* value of a conjunctive rule  $conj$  is defined on the product of the *MF* value of its selectors, adjusted by the rule's weight in the training set:

$$MF(conj, e) = \prod_k MF(sel_k, e) \times \frac{n(conj)}{N} \quad (6.4)$$

where  $n(conj)$  is the number of examples in the training set that are covered by the  $conj$  rule, and  $N$  is the total number of examples in the training set.

The *MF* of a class  $c_i$  is the probabilistic sum of all the conjunctive rules for the class. In the case of two rules,  $conj_1$  and  $conj_2$ , it is given by the following formula:

$$MF(c_i, e) = MF(conj_1, e) + MF(conj_2) - MF(conj_1, e)MF(conj_2, e). \quad (6.5)$$

If there are more than two rules for the class, we use this formula recursively.

The measure of fit method interprets the *MF* value as the *closeness* of the example to the class, and chooses the class  $c_i$ , which maximizes  $MF(c_i, e)$  as  $e$ 's classification.

**Multiple Match.** Multiple match is caused by overgeneralization of the training examples at induction time. All rule induction algorithms implement

generalization and specialization, explicitly or implicitly. We try to generalize the positive examples as far as we can until it covers negative examples. Once negative examples are covered, we need to specialize the concept description to exclude them. However, because the training set is generally incomplete, we need to carry out generalization and specialization under the closed world assumption. A missing example, say  $(X_1 = a, X_2 = 1)$ , can be assumed to belong to a concept  $c$  when we have found no negative examples of the concept that take the value of  $a$  on  $X_1$ . It can also be assumed to be a negative example at the same induction process if we find that all the negative examples and no positive examples in the training set have the property of  $X_2 = 1$ . The example in this case could well be covered by the descriptions of both the concept and another concept (which might be something like “not  $c$ ”), and therefore a multiple match happens in the subsequent deduction when the example appears in the test set.

Multiple match resolution needs to provide some measurements or criteria to judge which class is closer or more reliable to the test example in question.

- First hit. The simplest way to solve the multiple match issue is to use the first rule that classifies the example to determine the example’s classification. If the rules from induction have been sorted and ordered according to their reliability or their class reliability (e.g., putting rules related to the largest class before others), this simple method can be expected to produce reasonable results.

The advantage of this method is that it is straightforward and efficient in execution time. However, the price for the efficiency at deduction time is that the rules from induction need to be sorted.

- Largest class. The largest class method to solve multiple match examples has the same rationale as is applied to the no match case. It counts the coverage of those classes in the training set that are applicable to the test example in question, and then selects the largest one.

This method can sometimes produce good results, but because it does not use any information about the structure of the problem domain, it is not always reliable.

- Largest rule. Instead of using the coverage of the rules for a class, we can use the coverage of each conjunctive rule. However, the analysis with the largest class method also applies to this method.

- Estimate of probability. The *estimate of probability* [Michalski et al. 86] for handling the multiple match case assigns an  $EP$  value to every class by examining the size of the satisfied conjunctive rules.

The  $EP$  value for a conjunctive rule  $conj$  that is satisfied by an example  $e$  is defined as

$$EP(conj, e) = \begin{cases} \frac{n(conj)}{N} & \text{if } conj \text{ is satisfied by } e \\ 0 & \text{otherwise} \end{cases} \quad (6.6)$$

where  $n(conj)$  is the weight of  $conj$  and  $N$  is the number of examples in the training set.

The  $EP$  value of a class  $c_i$  is defined as the probabilistic sum of all the conjunctive rules for the class. In the case of two rules,  $conj_1$  and  $conj_2$ , it is given by the following formula

$$EP(c_i, e) = EP(conj_1, e) + EP(conj_2) - EP(conj_1, e)EP(conj_2, e). \quad (6.7)$$

If there are more than two rules for the class, we use the formula recursively.

The estimate of probability method chooses the class with the highest  $EP$  value to classify the example.

### 6.3 DEALING WITH REAL-VALUED ATTRIBUTES

Many inductive algorithms have been able to produce remarkably accurate rules even for difficult problem domains, in some cases performing as well as or better than human experts in the field. However, when the problem contains attributes that take values from continuous domains (i.e., real numbers or integers), their performance decreases in terms of accuracy. Although it is understandable that statistical methods (such as the MML method [Wallace & Bouton 68, Wallace & Freeman 92]) can achieve better results in numerical domains because they provide more powerful mechanisms to represent quantitative functions among numerical attributes or parameters, we need to deal with both symbolic and numerical domains in knowledge acquisition from realistic databases. There is no evidence to show that statistical methods outperform decision tree construction methods or rule induction in symbolic domains.

In the context of rule induction, dealing with a continuous domain means discretization of the numerical domain into a certain number of intervals. The discretized intervals can be treated in a similar way to nominal values during induction and deduction.

The most difficult aspect of discretization is to find the right places to set up interval borders. This section outlines some typical discretization methods. In the following account,  $N$  indicates the number of examples in the training set,  $m$  is the number of classes or concepts that these examples are classified into,  $a$  is the number of attributes used to describe each example, and  $d$  will be the number of intervals generated by discretization. In all these methods, sorting the values of the continuous attribute in question in ascending order is always useful before discretization is carried out.

### 6.3.1 The Simplest Class-Separating Method

The simplest discretization method is to place interval borders between each adjacent pair of examples that are not classified into the same class. Suppose the pair of adjacent values on attribute  $X$  are  $x_1$  and  $x_2$ ,  $x = (x_1 + x_2)/2$  can be taken as an interval border.

If the continuous attribute in question is very informative, which means that positive and negative examples take different value intervals on the attribute, this method is very efficient and useful. You can find, for example, that professors and lecturers at Australian universities have distinctive salary ranges, and the continuous attribute salary is very informative in distinguishing academic positions. However, this method tends to produce too many intervals on those attributes that are not very informative. These intervals can also easily confuse algorithms like HCV because a 0.1<sup>6</sup> difference between a positive example and a negative one on a numerical attribute makes one more interval.

### 6.3.2 Bayesian Classifiers

According to Bayes formula:

$$P(c_j|x) = \frac{P(x|c_j)P(c_j)}{\sum_{k=1}^m P(x|c_k)P(c_k)} \quad (6.8)$$

where  $P(c_j|x)$  is the probability of an example belonging to class  $c_j$  if the example takes value  $x$  on the continuous attribute in question,  $P(x|c_j)$  is the probability of the example taking value  $x$  on the attribute if it is classified in the class  $c_j$ .

$P(c_j)$  can be approximated by using one of the three probability estimation methods mentioned in Section 6.2.3, and  $P(c_j|x)$  can take the frequency

of  $c_j$  under  $x$  over all the examples in the training set.

Given  $P(c_j)$  and  $P(c_j|x)$ , we can construct a probability curve:

$$f_j(x) = P(x|c_j)P(c_j) \quad (6.9)$$

for each class  $c_j$ . When the curves for every class have been constructed, interval borders are placed on each of those points where the leading curves are different on its two sides. Between each pair of those points including the two open ends,  $-\infty$  and  $+\infty$ , the leading curve is the same.

We call a discretization implemented by this method a Bayesian classifier [Hong 94].

### 6.3.3 The Information Gain Heuristic

When the examples in the training set have taken values of  $x_1, \dots, x_n$  in ascending order on a continuous attribute, we can use the information gain heuristic adopted in ID3 (see Chapter 4) to find a most informative border to split the value domain of the continuous attribute. [Fayyad & Irani 92] showed that the maximum information gain by the heuristic is always achieved at a cut point (say, the midpoint) between the values taken by two examples of different classes.

We can adopt the information gain heuristic in the following way. Each  $x = (x_i + x_{i+1})/2$  ( $i = 1, \dots, n - 1$ ) is a possible cut point if  $x_i$  and  $x_{i+1}$  have been taken by examples of different classes in the training set. Use the information gain heuristic to check each of the possible cut points and find the best split point. Run the same process on the left and right halves of the splitting to split them further. The number of intervals produced this way may be very large if the attribute is not very informative. [Catlett 91] proposed some criteria to stop the recursive splitting:

- Stop if the information gain on all cut points is the same.
- Stop if the number of examples to split is less than a certain number (e.g., 14).
- Limit the number of intervals to be produced to a certain number (e.g., 8).

In C4.5 [Quinlan 93], the information gain approach is revised in the following ways. First, each of the possible cut points is not the midpoint between the two nearest values, but rather the greatest value in the entire

training set that does not exceed the midpoint. This ensures that all border values occur in the training data. Each border value in this case is not necessarily the same as the lower of the two neighboring values because all training examples are examined for the selection. Second, C4.5 adopts the information gain ratio (see Section 4.5.7 of Chapter 4) rather than the information gain heuristic. Finally, C4.5 does binarization of continuous attributes, which means only one interval border is found for each continuous attribute.

### 6.3.4 Other Methods

In addition to the methods already mentioned, the HCV (Version 2.0) software (see Appendix A) has implemented a few other simple methods, such as the equal distance division, grouping, and the  $k$ -nearest neighbors. Among others, the way to allow the user to specify their own discretization in the input files is very useful for integrating domain knowledge.

## 6.4 FUZZY INTERPRETATION OF DISCRETIZED INTERVALS

Once each continuous domain has been discretized into intervals, we can treat the intervals as discrete values in induction and deduction. This is the normal way that most existing systems have taken.

However, discretization of continuous domains does not always fit accurate interpretation. To say an age greater than 50 is old or a temperature above 32 is high is fuzzy. In these kinds of cases, fuzzy interpretation of the discretized intervals at deduction time could be very valuable. Rather than taking the chosen split points mentioned in Section 6.3 as sharp borders, we can instead place some kind of curve at each cut point as fuzzy borders. With these fuzzy borders, a value can be classified into a few different intervals at the same time, with varying degrees. This could change a single match case to a multiple match, and a no match case to a single or even multiple match. Deduction with fuzzy borders of discretized intervals is called *fuzzy matching*. In the multiple match case, we take the interval with the greatest degree as the value's discrete value.

Some fuzzy methods have been tried for deduction in the HCV (Version 2.0) software with numerical attributes. Although their results are significant when combined with other deduction methods [Wu & Måhlén 95],

fuzzy methods have not contributed as much as one can expect to the accuracy of deduction on their own. This is likely because fuzziness is strongly domain dependent. We need to incorporate domain information when defining the fuzzy borders if we would like to achieve significant results with specific domains.