Handling Unknown and Imprecise Attribute Values in Propositional Rule Learning: A Feature-Based Approach

Dragan Gamberger, Nada Lavrač, Johannes Fürnkranz

July 22, 2008

Abstract

Rule learning systems use features as the main building blocks for rules. A feature can be a simple attribute-value test or a test of the validity of a complex domain knowledge relationship. Most existing concept learning systems generate features in the rule construction process. However, the separation of feature generation and rule construction processes has several theoretical and practical advantages. In particular, the proposed transformation from the attribute to the feature space motivates a novel, theoretically justified procedure for handling of unknown attribute values. This approach suggests also a novel procedure for handling imprecision of numerical attributes. The possibility of controlling the expected imprecision of numerical attributes during the induction process is a novel machine learning concept which has a high application potential for solving real world problems.

Keywords: rule learning, features, unknown attribute value, imprecision of attribute values

1 Introduction

All real world applications of inductive learning systems are confronted with the problem of unknown attribute values in the training set. The simplest approach is to ignore examples with unknown attribute values. But this approach can be applied only when the number of unknown values is small and the number of available examples is very large. A commonly used approach is to replace missing values with a default value in the data preparation phase. For example, CN2 [2] replaces unknown values of discrete attributes by most commonly occurring value (the *mode* value), and unknown values for continuous attributes by the average value (the *mean* value). It is also possible to substitute the unknown values by random values or to try to estimate their values from known values of other attributes in the same example. These and similar techniques have drawbacks, especially when the number of unknown attribute values is high.

Extensive experiments presented in [1] demonstrate that none of the mentioned approaches is absolutely superior compared to the others.

An alternative approach suggested in [9] is to *reduce the apparent gain from testing attribute A by the proportion of cases with unknown values of A*. The rationale behind this approach is that testing attribute A will yield no information when it has unknown values, and when A has unknown values for all examples it is completely useless. The advantage of the approach is that it is theoretically sound. The problem is that its implementation is not a simple task. In decision tree induction we can easily appropriately reduce the information gain for the node testing attribute A that includes unknown values, but we do not have an effective method to partition examples with unknown values based on such a node. The consequence is that we can not expect optimal performance in nodes below the one that is based on an attribute that has many unknown values. In decision tree induction experiments reported in [9] the approach had similar prediction quality as those based on substituting unknown values by some known value in data preprocessing.

The approach based on reducing the gain from testing attributes with unknown values seems much more appropriate for covering rule induction approaches. The reason is that we do not have to partition the training set as in decision tree learning, but only to appropriately redefine the used heuristic evaluation measure so that the resulting value will get reduced by the proportion of unknown values. Although the principle is simple, there is a practical problem that, in contrast to decision tree induction, rules typically represent simultaneous decisions based on more than one attribute value and this makes the necessary computations very complex. To the best of our knowledge there is no rule inductive system that implements this approach. Even in [1] where very different approaches for handling unknown values were tested in the rule learning setting, the approach based on reducing the evaluation quality values for unknown attribute values was not mentioned.

In this paper we present a simple and straightforward approach to handling of unknown values in a rule learning setting using covering rule evaluation measures. It is based on a possibility to separate the feature construction process from the rule construction process. In the first phase we construct all potentially useful features and construct covering tables with true and false elements describing covering properties of features on all the training examples. In the second phase, representing the actual rule construction process, we use the information from these covering tables to find combinations of features with optimal covering properties. The problem of unknown attribute values is solved during the covering table construction. Covering values true and false are set so that covering quality of features is reduced always when the corresponding attribute value is unknown. After that, the second phase is executed in the same way as if all attribute values had known values. It means that we do not need to modify the used heuristic evaluation measures at all. The consequence of appropriately set covering values for features based on unknown attribute values is that standard covering evaluation measures will result in the adequately reduced feature and rule quality. The approach is applicable regardless of the used covering

Table 1: Illustration of simple and complex features in a domain with three examples described by three continuous attributes.

Attributes			Features	
A_i	A_j	A_k	$A_i > 3$	$A_k < 2 \cdot (A_j - A_i)$
7	1.5	2	*true*	*false*
4	3	-4	*true*	*true*
1.07	2	0	*false*	*true*

evaluation measure but it is not effective for systems using information gain evaluation measures.

The organisation of the paper is as follows. In Section 2 we introduce the concept of features, we briefly describe an algorithm for the construction of simple features, and illustrate the covering tables construction process. Using this framework we present the novel approach for handling unknown attribute values in Section 3. In Section 4 we exploit the proposed unknown value handling methodology also in handling the imprecision of continuous (numerical) attributes. Finally we describe how the same approach can be applied on ordered nominal attributes with application in DNA gene expression data analysis.

2 Features in propositional rule learning

Features describe properties of examples (instances). An example either has the property or it does not have this property. Thus, features are always Boolean-valued, i.e., either *true* or *false*. Features can be simple literals that test a value of a single attribute, like $A_i > 3$, or they can represent complex logical and numerical relations, integrating properties of multiple attributes, like $A_k < 2 \cdot (A_j - A_i)$, as illustrated in Table 1.

It is important to realize that features differ from the attributes that describe instances in the input data. Attributes can be numerical variables (with values like 7 or 1.5) or nominal or discrete variables (with values like *red* or *female*). In contrast with attributes, a feature cannot have a missing or unknown value. As a result, features are different from binary attributes even for binary-valued attributes that have values *true* and *false*.

Note that our use of the term *feature* is not fully aligned with the practice in the machine learning community where terms like *feature extraction*, *feature construction* or *feature selection* are used for approaches that aim at finding a suitable set of descriptors for the training examples by including expert knowledge, increasing the quality of learning or the expressiveness of the hypothesis language. As most learning algorithms, such as decision tree learners, focus on attributes, the term *feature* is frequently used as a synonym for *attribute*. In this paper, we clearly distinguish between these two terms.

Rule learning algorithms are *feature-based*, because rule learning algorithms

employ features as their basic building blocks, whereas, for example, decision tree learning algorithms are attribute-based, because decision trees are constructed from attributes. For many rule learning systems this is not obvious because the process of transformation of attributes into features is implicit and tightly integrated into the learning algorithm [2], [3]. In these systems, feature generation is part of the rule building process. The main reason for this strategy is the simplicity and especially the memory usage efficiency. Explicit usage of features requires that feature covering tables are constructed and these tables may be relatively large. Nevertheless, there are rule learning algorithms that explicitly construct covering tables before starting the rule construction process. A classical example is the LINUS system for converting relational learning problems into a propositional form [6]. In this framework, the concept of literal relevancy has been introduced by [7].

The generation of features for the given set of training examples is the first step in the rule learning process. It can also be viewed as the transformation from the attribute space into the space of features. Although generation of simple features is a straightforward and a rather simple task, generation of an appropriate set of sophisticated features is a hard task which is out of the scope of this paper.

2.1 Feature generation

An algorithm enabling the generation of simple features from attributes for a two-class classification problem is presented in [4]. The algorithm generates features separately and independently for each attribute. If an attribute is discrete then all distinct values in positive examples are detected and for them features of the form $Att = value$ are generated. Also all distinct values for negative examples are detected and from them features of the form $Att \neq value$ are generated. The number of generated features for a discrete attribute is equal to the sum of distinct attribute values occurring in positive and negative examples.

For a continuous attribute, features are generated in the following way: We identify pairs of neighboring values, where neighboring means that there is no other value between them. From these pairs, we compute the mean of the two neighboring values ($mean\_value$). If there are two neighboring values from different classes, then if the smaller of the two values is from the positive class we generate the feature $Att < mean\_value$, while if the smaller of the values is from the negative class we generate the feature $Att \geq mean\_value$. The number of features generated for a continuous attribute depends on the grouping of classes in the increasing value list but typically the number of generated features is proportional to the number of examples.

2.2 Covering tables

A *covering table* is a table which has examples as its rows and features as its columns. Table 2 presents a part of the covering table constructed for simple

Table 2: A part of the covering table generated for the domain with five examples and three attributes. Included are truth values for five out of ten features generated for this domain by the algorithm described in Section 2.1

Examples	Class	Attributes			Covering table					
Ex.	Cl.	A_1	A_2	A_3	$A_1 = red$	$A_1 \neq red$	$A_2 \geq 0.9$	$A_2 < 1.7$	$A_3 \geq 1.5$	...
p_1	$\oplus$	red	1.2	4	*true*	*false*	*true*	*true*	*true*	...
p_2	$\oplus$	blue	1.3	2	*false*	*true*	*true*	*true*	*true*	...
n_1	$\ominus$	green	2.1	3	*false*	*true*	*true*	*false*	*true*	...
n_2	$\ominus$	red	2.5	1	*true*	*false*	*true*	*false*	*false*	...
n_3	$\ominus$	green	0.6	1	*false*	*true*	*false*	*true*	*false*	...

features generated for a small domain with only three attributes. It can be noticed that the covering table has much more columns than the corresponding table presenting the examples by the attributes. Conversion from the attribute to the feature space thus presents a significant increase of the space complexity.

The covering table has only values *true* and *false* as its elements. These truth values represent the covering properties of features on the given set of examples. Together with example classes, the covering table is the basic information necessary for the rule induction process. Rule construction from the available set of features can be done by any covering based rule induction algorithm. The actual attribute values are no longer needed for rule construction. The significant difference of explicit feature generation compared to standard approaches is only that we do not generate features from attribute values during rule construction. Instead, the possible features with corresponding covering properties are obtained from the pre-prepared covering tables.

The price of explicit feature generation is the increase of the space complexity of rule learning algorithms. However, the advantages of explicit introduction of features are: a possibility to use the covering tables directly for rule construction, a possibility to introduce feature relevancy and ensure that only most relevant features really enter the rule learning process (described in [8]), and a possibility to solve problems of handling unknown attribute values and imprecision of continuous attributes during feature covering table construction.

3 Unknown attribute values

Note that a good feature to be used in rule construction has the property of being true for many positive examples and false for many negative examples. It is possible to formalize this statement in a form of a theorem. To do so, we start by defining the concept of p/n pairs of examples.

Definition: **p/n pair**
A p/n pair is a pair of training examples p_i/n_j where $p_i \in Pos$ and $n_j \in Neg$,

Table 3: A part of the covering table for the domain from Table 2 which includes a few unknown attribute values.

Examples	Class	Attributes			Covering table					
Ex.	Cl.	A_1	A_2	A_3	$A_1 = red$	$A_1 \neq red$	$A_2 \geq 0.9$	$A_2 < 1.7$	$A_3 \geq 1.5$	...
p_1	$\oplus$	red	**?**	4	*true*	*false*	**false**	**false**	*true*	...
p_2	$\oplus$	blue	1.3	2	*false*	*true*	*true*	*true*	*true*	...
n_1	$\ominus$	**?**	2.1	3	**true**	**true**	*true*	*false*	*true*	...
n_2	$\ominus$	red	2.5	**?**	*true*	*false*	*true*	*false*	**true**	...
n_3	$\ominus$	green	0.6	1	*false*	*true*	*false*	*true*	*false*	...

where Pos is the set of positive and Neg the set of negative examples.

Definition: **Discriminating feature**

Let F denote a set of features. Feature $f \in F$ discriminates a pair p_i/n_j iff feature f correctly classifies both examples, i.e., if feature f has value *true* for p_i and value *false* for n_j.

Theorem:

For training set E and set of features F a complete and consistent hypothesis H can be found using only features from set F if and only if for each possible p/n pair from training set E there exists at least one feature $f \in F$ that discriminates the p/n pair.

The proof of the theorem can be found in [7].

This theorem indicates that a good feature for rule construction is the one that discriminates many p/n pairs. An ideal feature has a property to be true for all positive and false for all negative examples because it discriminates all the p/n pairs. If the feature discriminates no p/n pairs it is completely irrelevant and can me immediately eliminated from the further rule construction process. In the situation when we have an example with the unknown attribute value, and when because of that we can not determine the real feature truth values for the features that test this attribute value, we should set the truth values of these features so that the features do not discriminate all p/n pairs built from the example. In this way, by reducing their covering properties, we directly penalize the features for those and only those examples that have unknown attribute values. If we have an attribute with unknown values for all the examples then all the features that test the attribute will not be able to discriminate any p/n pair and all such features will be irrelevant.

Definition 3 states that a feature discriminates a p/n pair only if the feature is true for the positive and false for the negative example. It means that when we have a positive example for which we can not determine the real feature truth values then we should set them to *false*. In this way the features will be not able to discriminate all p/n pairs built with this positive example. Based

on the same reasoning we see that for negative examples we should set feature truth values to value *true* when we do not know the real value. Only in this way we can ensure that the features will not discriminate all p/n pairs built with this negative example. The approach is illustrated in Table 3.

The consequence of the above procedure is that values *false* introduced in positive examples for unknown attribute values will be interpreted as false negative predictions by rule quality evaluation measures. In the same way, values *true* in negative examples will be interpreted as false positive predictions. Because every reasonable rule quality measure favours rules with many true positive and true negative classifications, the result is the degradation of the computed quality values for rules build from features that rely on attributes with unknown values.

The advantage of the proposed approach is that we do not have to change the rule induction process at all. Rules are constructed from completely specified features, rule quality measures may remain unchanged, and features can be combined in the rules in the same way as if all attribute values were known. Actually we can also incorporate the described approach in the rule induction algorithms that construct features during the rule construction process. But having explicit covering table is a good strategy not only because of handling unknown attribute values but also because it enables the detection and elimination of irrelevant features before the rule induction process starts.

At the end let us notice that the approach can be directly applied also for complex features. For example, feature $A_k < 2 \cdot (A_j - A_i)$ from Table 1 will have unknown value when any of attributes A_i, A_j, and A_k has unknown value. The principle is easily extendable to features of any complexity with included both numerical and logical operations.

4 Imprecision of continuous attributes

Another problem encountered in real world rule learning applications is the problem of imprecise values. Imprecision is inherent to most non-integer continuous attributes. There are two main reasons for the necessity of imprecision handling. The first is that some continuous attributes are not or can not be measured with high precision (like some biological properties) or their values are significantly fluctuating (e.g., human blood pressure). In such cases, if the example values are very near to the decision values used in the features then the actual feature truth values for such examples are unreliable. The second reason is that although some attributes like income or human age can be known very precisely, building rules from features that have supporting examples very near to the decision values used in the features may be a bad practice. The reasoning is that such features may lead to rules with significant overfitting, or, in case of descriptive induction, may result in non-intuitive rules like that 20 years old people have significantly different properties from those having 19 years and 11 months.

An approach to deal with inherent attribute imprecision, regardless of which

Table 4: A part of the covering table for the domain from Table 2 generated with the assumption of the expected attribute imprecision value $\delta = 0.35$.

Examples	Class	Attributes			Covering table					
Ex.	Cl.	A_1	A_2	A_3	$A_1 = red$	$A_1 \neq red$	$A_2 \geq 0.9$	$A_2 < 1.7$	$A_3 \geq 1.5$	...
p_1	$\oplus$	red	1.2	4	*true*	*false*	**false**	*true*	*true*	...
p_2	$\oplus$	blue	1.3	2	*false*	*true*	*true*	*true*	*true*	...
n_1	$\ominus$	green	2.1	3	*false*	*true*	*true*	*false*	*true*	...
n_2	$\ominus$	red	2.5	1	*true*	*false*	*true*	*false*	*false*	...
n_3	$\ominus$	green	0.6	1	*false*	*true*	**true**	*true*	*false*	...

type it is, is to treat attribute values near to the feature decision values as unknown values in order not to allow that such close values will affect feature quality. Such 'soft' unknown values are handled as standard unknown attribute values described in Section 3. It must be noted, however, that the actual attribute value is known, therefore it may happen that for a feature with some decision value it should be treated as unknown, while for some other feature with a different decision value it may be treated as a regular known value. More precisely, appropriate dealing with continuous attributes is such that in case of a feature decision value d we have to treat all attribute values v in the range $d-\delta < v < d+\delta$ as unknown attribute values, for δ being the user-defined attribute imprecision boundary.

For illustration in Table 4 we have assumed during covering table generation that the expected attribute imprecision value is 0.35 ($\delta = 0.35$). This assumption has practical consequences for the feature $A_2 \geq 0.9$ only, resulting by two 'soft' unknown values when attribute A_2 has values 1.2 and 0.6 which are less than δ far from the feature decision value 0.9. It can be noticed that this does not affect the truth values of the feature $A_2 < 1.7$ which has a different decision value.

The example presented in Table 5 demonstrates an artificial situation with two very similar continuous attributes such that from both ideal features discriminating all 25 p/n pairs can be generated. But when the notion of imprecision of continuous attributes is introduced, the situation may change significantly. The right part of the table presents covering properties of the same features but with imprecision of 0.17. Now none of the features is ideal. The first discriminates 9 and the second 16 p/n pairs. Although with assumed imprecision 0.0 both features seem equally good, with assumed imprecision 0.17 we have clear preference for the second feature. By analyzing the attribute values it really seems that it is better to use attribute A_2 because it may turn out to be a more reliable classifier in an imprecise environment.

It is important to notice that the presented approach to handling imprecision of continuous attributes can be applied also when we have ordered nominal

Table 5: The example demonstrates the difference in feature covering properties for imprecision values 0.0 and 0.17 respectively. Attributes A_1 and A_2 are very similar but there are significant differences in feature covering properties when different expected attribute imprecision is taken into account.

Examples		Attributes		Features (with $\delta = 0.0$)		Features (with $\delta = 0.17$)	
Ex.	Cl.	A_1	A_2	$A_1 < 1.95$	$A_2 < 1.95$	$A_1 < 1.95$	$A_2 < 1.95$
p_1	$\oplus$	1.5	1.5	*true*	*true*	*true*	*true*
p_2	$\oplus$	1.6	1.6	*true*	*true*	*true*	*true*
p_3	$\oplus$	1.7	1.65	*true*	*true*	*true*	*true*
p_4	$\oplus$	1.8	1.7	*true*	*true*	**false**	*true*
p_5	$\oplus$	1.9	1.8	*true*	*true*	**false**	**false**
n_1	$\ominus$	2.0	2.1	*false*	*false*	**true**	**true**
n_2	$\ominus$	2.1	2.2	*false*	*false*	**true**	*false*
n_3	$\ominus$	2.2	2.25	*false*	*false*	*false*	*false*
n_4	$\ominus$	2.3	2.3	*false*	*false*	*false*	*false*
n_5	$\ominus$	2.4	2.4	*false*	*false*	*false*	*false*

attributes. A good example are biological gene expression domains with presence call (signal specificity) values A (absent), P (present), and M (marginal). Although in this situation attributes have three discrete values A, M, and P, practically they are a formalization of values low, medium, and high. In each decision we have a problem to define if medium (marginal) values will be treated as low or high values. A principally clean solution is a possibility to treat medium values as unknown values. The described methodology can be interpreted also as handling imprecision of numerical attributes by selecting the expected imprecision value so that medium value M is always in the undefined region near to the assumed decision point. The methodology and the results are presented in [5].

5 Conclusions

We have defined features as the basic rule building blocks, and described a simple algorithm for the generation of propositional features that provably generates only the potentially relevant features both for discrete and continuous attributes. Explicit feature generation and presentation of their covering properties in the covering table has several important advantages. The basic idea is that the complete rule construction process can be done using only the information from the covering tables, which are an appropriate representation format for various learning algorithms.

The most relevant consequence is a possibility of systematic handling of relevancy of features with the possibility to detect and eliminate irrelevant features from the process of classification rule learning. In this paper we have demonstrated that explicit definition of features is useful also for systematic handling of

unknown attribute values. In contrast to existing approaches which try to substitute an unknown value with some good approximation, we have addressed the problem of handling of unknown values by appropriately defining feature truth values for such attribute values. The approach is simple, applicable also to complex features based on many attributes, and well justified.

The problem of imprecision of continuous attributes is seldom analyzed in the rule learning framework although it can be very relevant for real life domains. The problem is solved so that attribute values near to the feature decision value are treated as unknown attribute values. The approach relies on the user's estimation of the expected imprecision levels and the possibility to efficiently handle unknown attribute values through covering properties of generated features. There is a possibility to apply the same approach also for nominal values representing classes of numerical values which can be ordered by magnitude.

Signal specificity attributes in gene expression domains with values absent, marginal, and present are a good example of such attribute. It is known that gene expression domains are very prone to overfitting due to a large number of attributes in domains with a very modest number of examples. Extensive experiments in these domains have demonstrated that using signal specificity instead of signal intensity values, especially in combination by handling marginal values as unknown values is a effective method for overfitting prevention [5]. Also, we have applied the described approach for handling imprecision of numerical attributes in different, especially medical domains [4]. By experimenting with various expected imprecision levels in all of these domains the approach has demonstrated to be able to help construct features and rules describing general concepts easily interpretable by humans.

References

[1] I. Bruha and F. Franek. Comparison of various routines for unknown attribute value processing: The covering paradigm. *International Journal of Pattern Recognition and Artificial Intelligence*, 10(8):939–955, 1996.

[2] P. Clark and T. Niblett. The CN2 induction algorithm. *Machine Learning*, 3(4):261–283, 1989.

[3] W.W. Cohen. Fast effective rule induction. In A. Prieditis and S. Russell, editors, *Proceedings of the 12th International Conference on Machine Learning (ICML-95)*, pp. 115–123, Morgan Kaufmann, 1995.

[4] D. Gamberger and N. Lavrač. Expert-guided subgroup discovery: Methodology and application. *Journal of Artificial Intelligence Research*, 17:501–527, 2002.

[5] D. Gamberger, N. Lavrač, F. Zelezny, and J. Tolar. Induction of comprehensible models for gene expression datasets by subgroup discovery methodology. *Journal of Biomedical Informatics*, 37(4):269–284, 2004.

[6] N. Lavrač and S. Džeroski. *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, 1994.

[7] N. Lavrač, D. Gamberger, and V. Jovanoski. A study of relevance for learning in deductive databases. *Journal of Logic Programming*, 40(2/3): 215–249, 1999.

[8] N. Lavrač and D. Gamberger. Relevancy in constraint-based subgroup discovery. In JF. Boulicaut, L. De Raedt, H. Mannila, editors, *Constraint-Based Mining and Inductive Databases*. Springer, pp. 243–266 2005.

[9] J.R. Quinlan. Unknown Attribute Values in Induction. *In Proceedings of the 6th International Machine Learning Workshop, ML-1989*, pp. 164–168, 1989.