Fuzzy Interpretation of Induction Results

Xindong Wu and Petter Måhlén[†]

Department of Software Development, Monash University
900 Dandenong Road, Melbourne, VIC 3145, Australia
xindong@insect.sd.monash.edu.au

[†] Department of Numerical Analysis and Computer Science
Royal Institute of Technology, Stockholm, Sweden

Abstract

When applying rules induced from training examples to a test example, there are three possible cases which demand different actions: (1) *no match*, (2) *single match*, and (3) *multiple match*. Existing techniques for dealing with the first and third cases are exclusively based on probability estimation. However, when there are continuous attributes in the example space, and if these attributes have been discretised into intervals before induction, fuzzy interpretation of the discretised intervals at deduction time could be very valuable. This paper introduces the idea of using fuzzy borders for interpretation of discretised intervals at deduction time, and outlines the results we have obtained with the HCV (Version 2.0) software.

Introduction

Knowledge discovery in databases (KDD) is a research frontier (Wu 93a) for both database technology and machine learning techniques, and has seen sustained research over recent years. It acts as a link between the two fields, thus offering a dual benefit. Firstly, since database technology has already found wide application in many fields, machine learning research obviously stands to gain from this greater exposure and established technological foundation. Secondly, as databases grow in both number and size, the prospect of mining them for new, useful knowledge becomes yet more enticing. Machine learning techniques can augment the ability of existing DBMSs to represent, acquire, and process a collection of expertise such as those which form part of the semantics of many advanced applications.

Generally speaking, all kinds of attribute-based learning algorithms can be adapted to extract knowledge from databases. It is not difficult to add an induction engine to an existing database system in an *ad hoc* way to implement rule induction from databases or design some specific engines to learn from domain-specific data sets. However, when we integrate machine learning techniques into database systems, we must face many problems such as:

- Efficient induction algorithms are needed. The algorithms should be capable of being applied to realistic databases, *e.g.* $\geq 10^6$ relational tuples. Exponential or even medium-order polynomial complexity will not be of practical use.
- The knowledge learned needs to be tested and/or used back in the learning systems.
- Noise (including missing information) has to be effectively handled. Machine learning is different from mathematic induction. We cannot assume that the data in the given databases is complete. There are various sources of noise including missing values in real-world databases. To produce acceptable results for realistic applications, noise handling facilities are often essential to learning algorithms.
- Numerical data and symbolic data are equally important in practical application. Existing learning algorithms can be generally divided into two groups: numerical methods including statistical methods and neural networks which are good at processing numerical data in noisy environments, and symbolic AI methods which are more efficient in dealing with symbolic or nominal data. It has been a long term dispute that AI methods (especially the decision trees) are too simple to represent the real world. In the meanwhile, we can also easily argue that numerical methods are not good enough to represent and manipulate logic relationships among symbolic values. We need to have induction algorithms which can effectively deal with both types of data.

There are quite a few induction algorithms such as the ID3-like algorithms (Quinlan 86; Quinlan 93) and HCV (Wu 93b; Wu 95) which are low-order polynomial in both time and space. However, since induction from databases relies to a great extent on the quality of the training databases, interpreting induction results (say, rules) to classify a new example needs to face three possible cases which demand different actions:

- *No match*: No rules match the example;
- *Single match*: One or more rules indicate the same class match, and
- *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 rules, the rules will face the same problems (Quinlan 87; Quinlan 93).

In the single match case, the choice of class to the example is naturally the class indicated by the rules. Existing techniques for dealing with the first and third cases (Wu 95) are exclusively based on probability estimation. Among them, the Measure of Fit for dealing with the no match case and the Estimate of Probability for handling the multiple match case developed in (Michalski *et al.* 86) have been widely adopted in the KDD community.

The Measure of Fit and Estimate of Probability methods perform quite well with problem domains where no real-valued attributes are involved. However, when a problem contains attributes that take values from continuous domains (i.e. real numbers or integers), the performance of both methods, especially in terms of accuracy, decreases. In existing induction algorithms, dealing with continuous domains is based on discretisation of them into a certain number of intervals. There are quite a few strategies available for discretisation, such as (Wu 95) Bayesian classifiers and the information gain method. Once each continuous domain has been discretised into intervals, the intervals are treated as discrete values in induction and deduction. This is the standard way all existing systems have taken. However, discretisation of continuous domains does not always fit accurate interpretation. To say an age greater than 50 is *old* or a temperature above 32 centigrades is *high* is fuzzy. In these kinds of cases, fuzzy interpretation of the discretised intervals at deduction time could be very valuable. Rather than taking the cut points decided by the discretisation methods 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 discretised intervals is called *fuzzy matching*. In the multiple match case, we can take the interval with the greatest degree as the value's discrete value.

Discretisation of Continuous Attributes

When there are both symbolic and continuous attributes in an example set for induction, the standard approach is discretise the numerical domains of these continuous attributes into a certain number of intervals. The discretised intervals can be treated in a similar way to nominal values during induction and deduction.

The most difficult aspect of discretisation is to find the right places to set up interval borders. This section reviews some typical discretisation methods. In the following account, N indicates the number of examples in the training set, c is the number of classes or concepts that these examples are classified into, and d is the number of intervals generated by discretisation. In all these methods, sorting the values of the continuous attribute in question in ascending order is always useful before discretisation is carried out.

The simplest class-separating method

The simplest discretisation 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 which are not very informative. These intervals can also easily confuse algorithms like HCV (Wu 93b) because a 0.1^6 difference between a positive example and a negative one on a numerical attribute makes one more interval. The worst case is $d = N - 1$, where a border has to be set up between every pair of examples.

Bayesian classifiers

According to Bayes formula,

$$P(c_j|x) = \frac{P(x|c_j)P(c_j)}{\sum_{k=1}^{c} P(x|c_k)P(c_k)} \qquad (1)$$

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 following three probability estimation methods: relative frequency, Laplacian Law of Succession (Niblett and Bratko 87), or the m-estimate (Lavrac & Dzeroski 94), 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,

$$f_j(x) = P(x|c_j)P(c_j) \quad (2)$$

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 discretisation implemented by the above method a Bayesian classifier (Hong 94).

The information gain heuristic

When the examples in the training set have taken values of $x_1, ..., x_n$ in ascending order on a continuous attribute, we can use the information gain heuristic adopted in ID3 (Quinlan 86) to find a most informative border to split the value domain of the continuous attribute. (Fayyad & Irani 92) has shown that the maximum information gain by the heuristic is always achieved at a cut point (say, the mid-point) between the values taken by two examples of different classes.

We can adopt the information gain heuristic in the following way. Each $x = (x_i + x_{i+1})/2$ $(i = 1, ..., n-1)$ 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) has 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.* fourteen), and
- Limit the number of intervals to be produced to a certain number (*e.g.* eight).

In C4.5 (Quinlan 93), the information gain approach is revised in the following ways. Firstly, 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 neighbouring values since all training examples are examined for the selection. Secondly, C4.5 (Quinlan 93) adopts the information gain ratio 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.

Fuzzy Borders and Fuzzy Interpretation

Rather than taking the cut points set up by discretisation methods as sharp borders, each interval is associated with a specific membership function with fuzzy methods. The membership function measures the degree of a value belonging to the interval. In fact, sharp intervals can be treated as a special case of fuzzy borders: the membership function for an interval with sharp borders takes value 1 iff the value is inside the interval and 0 otherwise, and one value can belong to one interval only. Figure 1 shows the difference between sharp borders and fuzzy ones.

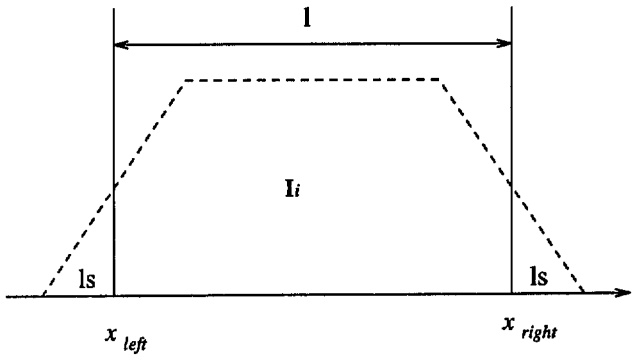

Figure 1: *Sharp and Fuzzy Borders*

In Figure 1, x_{left} and x_{right} are the left and right sharp borders of interval I_i respectively, and $l = x_{right} - x_{left}$ is the original length of the interval. s is the spread parameter, which indicates the length that an interval should be extended at each end. When the parameter is 0.1, for example, the interval in Figure 1 spreads out into adjacent intervals for twenty percent of its original length. In HCV (Version 2.0) (Wu *et al* 95), s is a user-specified parameter with default being 0.1.

The match of an example taking value x on a specific attribute domain with an interval is defined as the value of the membership function of the interval calculated for x.

We have implemented three membership functions (see Appendix) in HCV (Version 2.0), with which two methods for calculating the match degree of a value x with a selector[1] or conjunction have been implemented. The first takes the maximum membership degree of the value in all of the intervals involved in the selector. The drawback of this method is that if two adjacent intervals belong to the same selector, a value close to the border between the two intervals will get a very low membership value in both, leading to a low overall membership degree even if it is well covered by the selector. In an attempt to remedy this, the other method adds with fuzzy plus[2] all the fuzzy membership degrees within one selector.

The HCV (Version 2.0) software

The HCV algorithm (Wu 93b) is a representative of the extension matrix based family of attribute-based induction algorithms, originating with J.R. Hong's AE1 (Hong 85). By dividing the positive examples (PE) of a specific concept in a given example set into intersecting groups and adopting a set of strategies to find a heuristic conjunctive rule in each group which covers all the group's positive examples and none of the negative examples (NE), HCV can find a rule in the form of variable-valued logic for the concept in low-order polynomial time. If there exists at least one conjunctive rule in a given training example set for PE against NE, the rule produced by HCV must be a conjunctive one. The rules in variable-valued logic generated by HCV have been shown empirically to be more compact than the decision trees or their equivalent decision rules produced by the ID3 algorithm (the best-known induction algorithm to date) and its successors (e.g., C4.5) in terms of the numbers of conjunctive rules and conjunctions.

The HCV (Version 2.0) software is a C++ implementation of the HCV algorithm. In this implementation, HCV can work with noisy and real-valued domains as well as nominal and noise-free databases. It also provides a set of deduction facilities for the user to test the accuracy of the produced rules on test examples. The detailed description of the software is included in (Wu *et al* 95).

In addition to a set of discretisation facilities, such as the Bayesian classifiers and the information gain heuristic, and the fuzzy borders mentioned above, HCV (Version 2.0) permits the user to specify their own discretisation of real-valued attributes by providing a set of intervals in the structure file, which specifies the attributes (with their order and value domains) and classes used in the data files. This is a very useful way for integrating domain information.

Hybrid Interpretation

Extensive experiments have been carried out with the above fuzzy methods in HCV (Version 2.0) on a large set of databases from the University of California at Irvine Repository of Machine Learning Databases. However, the results were much less encouraging than what we expected when we were trying to justify that fuzzy borders are generally more reliable than sharp borders with numerical domains.

We have analysed the results by fuzzy methods and those with sharp borders, and found that the accuracy of the single matches is in general much better than no matches and multiple matches with all methods. With the multiple match case, the *Estimate of Probability* (Michalski *et al.* 86) with the Laplacian Law of Succession (Niblett and Bratko 87) outperforms other methods including fuzzy matching. These observations motivated the development of a hybrid interpretation in HCV (Version 2.0) with fuzzy matching and the Estimate of Probability.

The hybrid method works as follows. In the single match case, we do not provide any probability analysis or fuzzy borders. In the multiple match case, the Estimate of Probability method with sharp borders is used to find the best class for the example in question. Only in the no match case, fuzzy borders are set up (with the polynomial membership function as default) in order to find a rule which is closest (with the maximum membership degree) to the example in question.

The hybrid method is an option for deduction in HCV (Version 2.0). The user can overrule it by specifying other methods (such as the combination of the Measure of Fit and the Estimate of Probability).

Conclusions

As mentioned above, fuzzy methods, although their results are significant when combined with other deduction methods, do not contribute as much as one can expect to the accuracy of deduction on their own. This is likely because all the experiments have not been specifically conducted with domain dependent information.

Fuzziness is strongly domain dependent. The HCV (Version 2.0) software has provided a way for the user to specify their own intervals and select their own fuzzy functions. This is an important direction to take if we

[1] A selector in variable-valued logic (Michalski 75) takes the general form

$$[X\#R]$$

where X is a variable or attribute, $\#$ is a relational operator (such as $=, \neq, <, >, \leq$, and $\geq$), and R is a list of one or more values (including discretised intervals) that X could take on.

[2] Fuzzy plus $\oplus$ is defined as follows: $a \oplus b = a + b - ab$.

would like to achieve significant results with specific domains.

References

J. Catlett, On Changing Continuous Attributes into Ordered Discrete Attributes, *Proceedings of EWSL-91*, 1991.

U.M. Fayyad and K.B. Irani, On the Handling of Continuous-Valued Attributes in Decision Tree Generation, *Machine Learning*, **8**(1992), 87–102.

J. Hong, AE1: An Extension Matrix Approximate Method for the General Covering Problem, *International Journal of Computer and Information Sciences*, **14**(1985), 6: 421–437.

J. R. Hong, *PKAS: A Practical Knowledge Acquisition System*, Unpublished (1994).

N. Lavrac and S. Dzeroski, *Inductive Logic Programming - Techniques and Applications*, Ellis Horwood, 1994.

R.S. Michalski, Variable-Valued Logic and Its Applications to Pattern Recognition and Machine Learning, *Computer Science and Multiple-Valued Logic Theory and Applications*, D.C. Rine (Ed.), Amsterdam: North-Holland, 1975, 506–534.

R.S. Michalski, I. Mozetic, J. Hong and N. Lavrac, The Multi-Purpose Incremental Learning System AQ15 and Its Testing Application to Three Medical Domains, *Proceedings of AAAI 1986*, 1986, 1041–1045.

T. Niblett and I. Bratko, *Learning Decision Rules in Noisy Domains*, Research and Development in Expert Systems III, M. A. Bramer (Ed.), Cambridge, New York Cambridge University Press, 1987 pp. 25–34

J.R. Quinlan, Induction of Decision Trees, *Machine Learning*, **1**(1986), 81–106.

J.R. Quinlan, Generating Production Rules from Decision Trees, *Proceedings of International Joint Conference on Artificial Intelligence*, J. McDermott (Ed.), Morgan Kaufmann Publishers, Inc., 1987, 304–307.

J.R. Quinlan, *C4.5: Programs for Machine Learning*, Morgan Kaufmann Publishers, 1992.

X. Wu, Inductive Learning: Algorithms and Frontiers, *Artificial Intelligence Review*, **7**(1993), 2: 93–108.

X. Wu, The HCV Induction Algorithm, *Proceedings of the 21st ACM Computer Science Conference*, S.C. Kwasny and J.F. Buck (Eds.), ACM Press, USA, 1993, 168–175.

X. Wu, *Knowledge Acquisition from Data Bases* (in press), 1995.

X. Wu, J. Krisár and P. Måhlén, *HCV (Version 2.0) User's Manual*, Department of Software Development, Monash University, Australia, 1995.

Appendix: Three Fuzzy Membership Functions Implemented in HCV (Version 2.0)

There are three functions in HCV (Version 2.0) which can be used to fuzzify interval borders.

The linear function (see Figure 2) is specified by

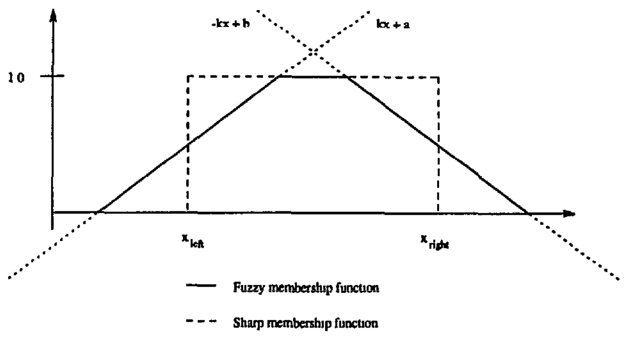

Figure 2: *The linear membership function*

the following expressions where s and l have the same meanings as in Figure 1.

$$k = \frac{1}{2sl}, \quad a = -kx_{left} + \frac{1}{2}, \quad b = kx_{right} + \frac{1}{2}$$

$$lin_{left}(x) = kx + a$$

$$lin_{right}(x) = -kx + b$$

$$lin(x) = MAX\{0, MIN\{1, lin_{left}(x), lin_{right}(x)\}\}$$

With the polynomial membership function (see Figure 3), the fuzzy borders are defined by a third-degree polynomial.

$$poly_{left}(x) = a_{left}x^3 + b_{left}x^2 + c_{left}x + d_{left}$$

$$poly_{right}(x) = a_{right}x^3 + b_{right}x^2 + c_{right}x + d_{right}$$

where

$$a_{left} = a_{right} = -\frac{1}{4(ls)^3}$$

$$b_{side} = -3a_{side}x_{side}$$

$$c_{side} = 3a_{side}(x_{side}^2 - (ls)^2)$$

$$d_{side} = -a(x_{side}^3 - 3x_{side}(ls)^2 + 2(ls)^3)$$

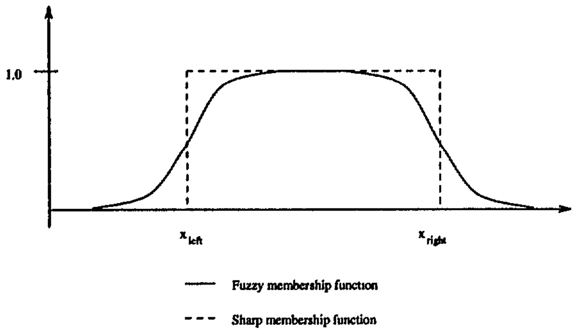

Figure 3: *The polynomial membership function*

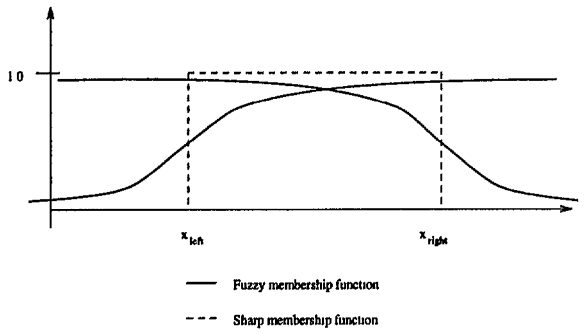

Figure 4: *The arctan membership function*

and $side \in \{left, right\}$, x_{side} is the sharp border on each side, and l and s are the original interval length and the spread respectively.

$$poly(x) = \begin{cases} poly_{left}(x) & if\ x_{left} - ls \le x \le x_{left} + ls \\ poly_{right}(x) & if\ x_{right} - ls \le x \le x_{right} + ls \\ 1 & if\ x_{left} + ls \le x \le x_{right} - ls \\ 0 & otherwise \end{cases}$$

The third membership function (see Figure 4) is the arctan function. The spread of the interval is used to indicate the *flatness* or linearity of the curve, and the fuzzy membership of an interval takes the minimum of the membership from the left and the one from the right. The function used to calculate the membership is:

$$arctan(x) = MIN\{\frac{1}{\pi}atan(\frac{x - x_{left}}{5s}) + \frac{1}{2},$$
$$\frac{-1}{\pi}atan(\frac{x - x_{right}}{5s}) + \frac{1}{2}\}.$$