ON CHANGING CONTINUOUS ATTRIBUTES INTO ORDERED DISCRETE ATTRIBUTES

J. CATLETT
Basser Department of Computer Science, University of Sydney, N.S.W. 2006, Australia
Email: jason@cs.su.oz.au

Abstract

The large real-world datasets now commonly tackled by machine learning algorithms are often described in terms of attributes whose values are real numbers on some continuous interval, rather than being taken from a small number of discrete values. Many algorithms are able to handle continuous attributes, but learning requires far more CPU time than for a corresponding task with discrete attributes. This paper describes how continuous attributes can be converted economically into ordered discrete attributes before being given to the learning system. Experimental results from a wide variety of domains suggest this change of representation does not often result in a significant loss of accuracy (in fact it sometimes significantly improves accuracy), but offers large reductions in learning time, typically more than a factor of 10 in domains with a large number of continuous attributes.

Keywords Discretisation, empirical concept learning, induction of decision trees

1. Introduction

As Subramanian (1989) concisely stated, "Present day learners depend on careful vocabulary engineering for their success." Many authors since Amarel (1968) have demonstrated the critical importance of the framework used to represent a given learning task, and the benefits to be gained from an appropriate change of representation. Recent work such as Rendell (1989) has advanced the state of the art of constructive induction, where the attributes given to a selective induction system such as ID3 (Quinlan 1979, 1983) are transformed into better attributes. Usually "better" is taken to mean "higher level", allowing the target concept to be expressed more concisely, making the product of the learning more accurate. The transformations often involve investigation of many complex combinations of the raw attributes, and can be computationally expensive, but

this work may be rewarded by error rates lower than could be obtained from induction directly on the raw attributes.

This paper too addresses the question of changing the representation of tasks given to an inductive learning system, but with the aim of lowering the computational cost of the learning rather than improving the accuracy of the product. The ideal representation would be one that allows the learner both to express the final concept most accurately (or concisely, or comprehensibly), and to compute that concept in the least time possible, but these two goals are in conflict. Thus the client for whom the learning is performed faces a trade-off between speed and accuracy. Changing the representation complicates this decision, because any change entails a computational cost.

The situations in which the client would want to reduce the cost of learning are becoming increasingly common as machine learning becomes a commercial technology. The main spur comes from problems of scale: training set of hundreds of thousands of instances are now being attacked (e.g. Sejnowski 1987), making learning time a limiting factor. As commercial knowledge acquisition tools that interface learning systems to corporate databases become popular, the management of the extraction and transformation of learning data will become a difficult logistic task, much of it needing to be transformed from numeric to symbolic form. Finally, in the past three years a class of applications that use incremental learning has come under the spotlight (for example, Utgoff 1988). Where this is done in real time with new examples becoming available during the induction, it may be advantageous to finish the induction earlier.

Various researchers have attacked the question of how to speed up the learning task. Selective induction algorithms such as ID3 that partition the instance space are generally very efficient and difficult to improve. Quinlan's first descriptions of ID3 (1979) included a method called windowing to speed up ID3 on large training sets, but Wirth & Catlett (1988) showed that on noisy data it usually costs rather than saves CPU time. Breiman, Friedman, Olshen & Stone (1984, pp 163-7) proposed a method of choosing the attribute to partition the space based on a random subset of the available instances, but this still retains much of the cost of dealing with continuous attributes.

The algorithm we describe for discretising a dataset could be used as before a variety of learning algorithms; here we use ID3, to provide a familiar basis for comparison. However, the rationale for some of the design decisions in the algorithm is based on the assumption that the learning system partitions the training set, and may not be appropriate for learners that build around a seed example, such as AQ15 (Michalski,

Mozetic, Hong, & Lavrac 1986).

ID3 is reviewed briefly in Section 2, with emphasis on continuous attributes. Other work related to continuous attributes is reviewed in Section 3. Section 4 introduces the algorithm for discretising continuous attributes. Section 5 describes domains used in experiments to evaluate this algorithm, and gives their results. We conclude with a statement of the areas to be explored further.

2. ID3

ID3 (Quinlan, 1979, 1983) is an inductive learning program that constructs classification rules in the form of decision trees. It is perhaps the most commonly found ML algorithm in both the scientific literature and in commercial systems; for illustrations of real-world applications see (Michie, 1987) or (Carter & Catlett, 1987). For a comprehensive introduction to the induction of decision trees see (Quinlan 1986). The following oversimplified summary aims merely to establish terminology and to emphasise some detail concerning continuous attributes.

The input to ID3 is a set of objects called the *training set.* The objects are described in terms of a fixed set of *attributes*, each attribute taking its value from a given set, and each object is assigned to one of a small fixed set of classes. The output is a decision tree that can be used to classify any object from that attribute space. Typically the tree has a low error rate on the training set, but is less accurate on unseen examples.

We can distinguish *continuous* attributes, whose values are real numbers on some interval, from *discrete* attributes, restricted to some finite (usually small) number of values. Discrete attributes can be further divided into *ordered* attributes, such as cold, tepid, warm, hot, or *unordered* attributes, such as Africa, America, Australia, Europe. Continuous attributes were not described in Quinlan's original ID3 paper (1979, 1983), nor were ordered discrete attributes, which were first reported in ASSISTANT (Kononenko, Bratko & Roskar, 1984). These simple extensions are now widespread in implementations.

ID3 builds its trees by progressively partitioning the training set: it chooses an attribute to split the set on, and then recursively builds a tree for each subset, until all the members of the subsets are of the same class. For each subset it must decide which is the best attribute to split on, which it does using an formula that assesses the gain in information theoretic terms of all possible splits. ID3 is a greedy algorithm with no lookahead, so making a bad choice of an attribute fragments the training set and reduces accuracy.

In choosing a continuous attribute to split the set, a *threshold* or *cutpoint* must also be selected. (These two terms are synonyms.) Computationally, this entails sorting the set on each attribute in turn, for each subset produced as the training set divides. Each sort takes time O($n$log$n$) where n is the number of elements in the subset.

When all the attributes are discrete (whether ordered or not) no sorting is required, merely space proportional to the number of values (which is quite small) and time O(n). Overall the algorithm takes time proportional to the product of the training set size, the number of attributes, and the size of the final tree. With continuous values the growth is superlinear, because sorting of the relevant subset of the training set must be done for every attribute at every node created during the building of the tree. With discretisation, sorting is done only once per attribute. As an illustration of the relative costs, Figure 1 shows for a single run the CPU time taken to build a tree from continuous versus discretised binary attributes (i.e. single threshold) for the waveform domain (described in the next section). The floating figures plot on a log scale the accuracy of the trees built from discretised data for various sized training sets. The corresponding accuracies for the original continuous data are not shown; they are approximately the same. The figures for CPU time are joined with a dashed line for the discretised version, and a solid line for the continuous version. The line marked linear is simply a straight line that approximates the early part of the continuous attributes curve, showing the latter to be superlinear.

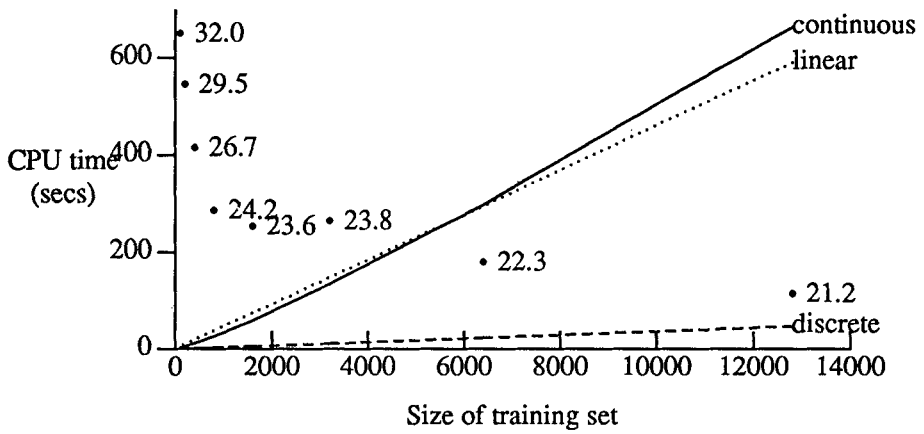

Figure 1. Learning time on discretised vs continuous attributes

This substantial difference in the CPU cost is the main motivation for changing continuous attributes into discrete ones. In the case of the wave domain illustrated above the learning time is reduced by a factor of about 14.

A further saving, in terms of space, comes from the fact that a floating point number typically takes four bytes to represent, while a discrete ordered attribute takes only one byte. (The implementation reported here reads in the data and converts it *in situ*, so space savings are not realised.) In very large domains (such the Heart domain, with 92 continuous attributes), memory requirements may be a limiting factor on the size of induction tasks that can be performed.

It may turn out that discretising attributes offers cognitive benefits as well as computations ones. In the thyroid domain, the discretising algorithm chose to a high degree of accuracy almost all the thresholds used by domain experts. Human experts may prefer to look at the set of thresholds chosen, and thereafter work from a tree labelled with symbolic values such as low, high, and very high, rather than using a tree with a large number of (sometimes very close) thresholds.

3. Relationship to other work on continuous attributes

The idea of discretising (or "quantizing") an attribute has appeared previously in the pattern recognition and statistics literature. (Wong & Chiu 1987) compare two techniques for discretising data. Their motivation for discretisation is not to reduce learning time, but to convert the real-valued data into a form suitable for their clustering algorithms. Their techniques do not require (or use) a distinguished class, as ID3 does. This would put them at a disadvantage in some applications, but makes them more widely applicable. Both techniques require the user to specify the number of intervals into which each attribute will be divided. Their first technique, called *equal width discretisation,* simply involves calculating the difference between the largest and smallest values for the attribute and placing the desired number of equal-width intervals between them. Their second technique, called *maximum marginal entropy discretisation,* requires sorting of the values, and basically allocates an equal number of examples to each interval. Where repeated values would cause that value to belong to more than one interval, the boundaries are adjusted so as to minimise an information theoretic measure on the discretised attribute. Note that this is not the same as ID3's measure because it is "class-blind"; it does not take into account the class of the examples, only the attribute values. For this reason we believe that such methods for discretisation may not be the best match for ID3; the results of experiments discussed at the end of Section 5 corroborate this.

Most of the commonly used additions to ID3, such as pruning and the treatment of unknown values (Quinlan 1989), are not affected by discretisation, but the

implementation of one less common addition, *soft thresholds*, requires care. Its motivation comes from a distaste for the "knife-edge" threshold used by ID3, that can make a small change in an attribute's value cause a large change in the class or class probability estimate given by the tree. A more "fuzzy" threshold has obvious appeal. A little inspection shows that the methods devised to do this are not excluded by discretisation, although choices arise which are not evaluated in this paper. The idea of soft thresholds was first suggested by Catlett to Carter, and details of Carter's implementation were published in (Carter & Catlett, 1987). The basic idea is choose in addition to each threshold, a subsidiary threshold either side of it; examples falling between these two thresholds are considered sufficiently close to warrant "fuzzy" treatment from both of the subtrees below that attribute choice. The method of choosing the two subsidiary thresholds was subsequently improved by (Quinlan 1987). When using discretisation, we could calculate the subsidiary thresholds at the time of discretisation. The alternative is to determine them after the tree is built. This would entail re-traversing it, splitting the training set according to the attribute at each node, and calculating the subsidiary thresholds in the normal manner. The cost of doing this would still be small compared to building the whole tree from undiscretised data.

4. Algorithm for discretising continuous attributes

This section describes the algorithm that chooses the thresholds. Although it can be thought of logically as a preprocessor to ID3, the most economical implementation would include it as part of the learning program because the setup chores of getting the data in place are the same, and most of the necessary routines are already in ID3.

The algorithm given here decides on the thresholds for each attribute without reference to the others. A more advanced version might take into account a situation where, for example, even the best threshold on an attribute gives a gain that is so poor compared to other attributes that it might be deemed not worth spending more CPU time on. That question is related to the task of eliminating irrelevant attributes, and has appeared in the constructive induction literature.

The first threshold is computed in the same way as ID3 chooses its thresholds. Thus we are guaranteed that the attribute and cutpoint of the root of the tree built will be the same in both the continuous and discrete versions. After this we face a question: whether to go on and choose another threshold, and if so, how to choose it.

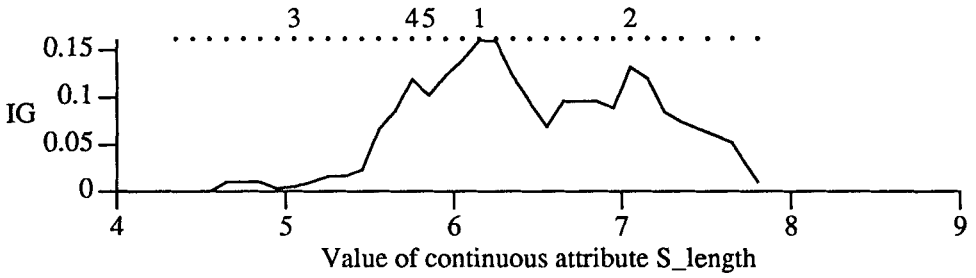

Figure 2. Initial information gain for Top cutpoints

Given that we are going to choose a second cutpoint, we must not make the obvious choice of the value with the second highest gain. Making this mistake typically results in a large number of cutpoints near the original one, offering very little additional discriminating power because the difference between them and the first cutpoint involves only a few examples. Instead we take the view that the situations where we will need the second cutpoint during the tree building occur where the set has already been split on the first cutpoint (or on some other attribute with a similar effect). Accordingly, we choose the subsequent cutpoints based on the subsets between the cutpoints already chosen. To illustrate this distinction, Figure 2 plots the information gain for all cutpoints on an attribute from the Iris domain. Each possible cutpoint is marked with a dot; the best five cutpoints are distinguished with a numeral above the dot denoting their rank. The third such cutpoint appears near-useless on the initial ranking, but after the population to the right of the best cutpoint is removed, it happens to become the best choice for the left subset.

Although the algorithms sounds like the same computation as goes on during the building of the tree, we are really avoiding the expensive O($n$log$n$) operation of resorting the values. This is because we select all the thresholds for an attribute in one step, rather than jumping back and forth. We merely have to adjust the counts to the left and right of the last threshold to remove the influence of the examples on the other side of the threshold. The procedure can then be applied recursively. This method has the reassuring property that for a learning task with just one continuous attribute the discretised version would yield the same tree as the the discretised form, provided that a sufficient number of thresholds is used.

As with all recursive procedures, there must be a condition to determine when to stop, when we have enough thresholds in an interval. This decision is very like the decision of when to stop splitting the training set when growing a tree, so techniques from ID3 such as the chi-squared cutoff (Quinlan 83) or Fisher's test (Quinlan 87) could be used.

Another possibility is to look at the information gain offered by the split versus the number of examples, and to use some statistical test that determines a confidence level that the gain is not due to chance. Finally, the simplest method is to set some crude absolute cutoffs, which is what was used to produce the figures shown in this paper. (For future reference we will call this version D-2, for Discretiser 2). But even the extremely crude criterion of "stop after the first threshold" (i.e. the "discretised" version of every continuous attributes is always binary) was enough to produce trees of approximately the same accuracy in two domains, Hyper and Wave.

The stopping condition of D-2 comprises the following four broad sub-conditions. (Following the goal of reporting experiments in sufficient detail to allow them to be reproduced, we give the specific parameters used in D-2. We are not claiming that these are the best such values, merely that they are reasonable.)

1. If the number of examples represented in the interval is sufficiently small, we should stop, because estimates made based on very small samples may not be reliable. The smallest number we could divide is two; D-2's cutoff is currently 14.

2. Given that our motivation is to reduce CPU time, it seems reasonable to place some overall maximum on the number of thresholds produced for any attribute. If this is not imposed, some larger domains, such as Heart, produce hundreds of thresholds on what looks like reasonable data to split, and could consume more time finding thresholds that will never be used than is saved by the conversion to discrete attributes. D-2 limits the number of thresholds (or actually the number that are likely to be produced by the depth-first recursion) to seven.

3. D-2 stops if the gain on all possible thresholds is equal. (This happens quite often with the value 0, suggesting that further splitting may be ineffective.) This presumption is theoretically unsound (it would be easy to contrive a domain where this costs accuracy), but in practice it usually means that further splitting on the attribute would be unfruitful.

4. If all the examples in an interval belong to the same class, there is no need to split it, so no further thresholds need be established. This sub-condition logically implies the previous condition, but the converse does not hold.

When the number of classes is greater than two, one more complication is necessary to stop thresholds that split off less frequent classes from being swamped by thresholds that discriminate well between the most frequent classes. Experimentation with some of the

thyroid domains, which contain some very unusual disorders, showed that this step was necessary. D-2 looks at an *n* class decision as *n* binary decisions by considering each class in turn is made the positive class with the others considered negative. At the end all the thresholds found are used. It would probably be possible to merge very close thresholds without loss of accuracy, but this was not tried; we doubt it would save very much time since an extra discrete value entails only a small amount of space and time.

5. Description of the domains and results of experiments

To test the effect of discretising on classification accuracy, we ran the following experiment on all the large domains with real valued attributes available to us. For each domain, we split the data randomly at least 20 times and, for each split, ran ID3 on the raw continuous data, then discretised the training and test data using thresholds calculated from the training data, and ran it on the discretised data. In both cases the trees were pruned on the training data using pessimistic pruning (Quinlan 1987). For each pair of data we recorded the original accuracy and the difference in accuracy due between the continuous and discrete data. The domains are described below.

1. Waveform: this was adapted from (Breiman et. al 1984). Although it is less popular than the discrete-valued faulty LED problem from the same book, it has been used in several comparison papers. Our implementation produces 42 real-valued attributes, half of them irrelevant but with the same distribution as the relevant ones. There are three classes, representing combinations of waveforms to which various kinds of noise is added to obtain the attributes.

2. Thyroid: this large and widely used dataset from the Garvan Institute in Sydney is described in (Quinlan, Compton, Horn & Lazarus 1988). It consists of case data and diagnoses for many disorders concerning thyroid hormones, of which the following were suitable for our experiments here (the number of classes is indicated in brackets): Hypo (5), Hyper (5), Binding Protein (3), Replacement (4) and Sick Euthyroid (2). The case data is described in terms of 29 attributes, of which six are real-valued: the patient's age and five assay readings. One unusual bonus offered by this domain is the knowledge of the actual thresholds used by specialists in interpreting the assays; almost all of them were found by D-2 (among many other thresholds).

3. Demon: this domain was deliberately contrived as the worst possible domain to give to this algorithm. Because the selection criterion looks at the relationship

between the mix of classes under its possible divisions, a division will appear useless if the subsets either side of a threshold have the same mix of classes. More formally, this occurs when the probability that an example's value for attribute A is above a threshold T is independent of its class:

$$P(A > T \mid C) = P(A > T)$$

Normally this is a good reason to ignore the attribute, but if we make its value effectively a "switch" on the thresholds relevant to other attributes, it becomes important to use it in combination with the other attributes. We chose three attributes A1 (the switch), A2, and A3, all with real values uniformly distributed on [0, 1], and the class being positive if the following expression hold true:

$$A1 < 0.5 \;\&\; A2 < 0.8 \;\&\; A3 > 0.2 \text{ or } A1 > 0.5 \;\&\; A2 > 0.2 \;\&\; A3 < 0.8$$

Since there is no noise in this domain ID3 finds a highly accurate tree after a few hundred training instances.

The ML literature has seen switching in parity checkers and multiplexers on boolean attributes (Wilson 1987), and the example of the boolean function A1 > A2 is often trotted out as an illustration of the importance of the correct formulation of attributes to ID3, but we know of no real domain where this sort of switching occurs naturally on real-valued attributes, let alone on an attribute that is perfectly independent of the class. This domain is of course very contrived; the point was to devise a worst case monster to pit the algorithm against.

4. Heart: this domain benefits most from discretising, because it consists of several thousand examples, each with 92 real-valued attributes, and takes about half an hour to produce a tree on a VAX-11/780. Discretising speeds up the induction by a factor of more than 50. The examples consist of measurements from heart patients over consecutive 30 second epochs, with the class indicating whether the heart was ischemic at that time. The application is described in (Oates, Cellar, Bernstein, Bailey & Freedman 1989).

5. Othello: the generator for this data was provided by Paul Utgoff, and is described briefly in (Utgoff 1988) and in more detail in (Utgoff & Heitman 1988). The examples represents successive board positions in a game of Othello, and are described in terms of 14 integer-values attributes (with only 29 distinct values per attribute). The class indicates whether the position described by by the first seven attributes can be shown to be better than the position described by the last seven.

The object is to be learn preference predicates.

Because the examples within a game are strongly related, it is in some ways fairer to take the test set from a different game. The results reported below call the latter "Othello B", and the usual practice of splitting a single uniform sample randomly into test and training sets "Othello A".

Table 1 shows the the results of the comparison experiments. The column headed "Error rate mean" gives the percentage error on the test set by ID3 for continuous attributes. The training set is also used to compute thresholds for converting the attributes to ordered discrete versions, and the difference in error rate between the continuous and discrete versions is given in the next column. For example, the first line shows that on the Iris database, the error rate was the same in the two versions, and the second line shows that for the Demon domain the discrete version averaged error rate more than 10 percentage points higher than the continuous version. This mean figure and standard deviation relate to the mean of the differences, not the difference of the means. The column headed *N* gives the number of trials. The column headed conclusion give the result of a two-sided t-test at the 5% level, with NSD denoting no significant difference (i.e. we accept the null hypothesis that the mean of the differences is zero, showing that there is no difference in error between the original continuous and the discretised version).

TABLE 1. Comparison experiments

Domain	Training set size	Test set size	Error rate mean	Difference of errors mean	Difference of errors stddev	Number of trials	Concl.
Iris	100	50	5.800	0.000	0.000	20	NSD
Demon	5000	3000	0.265	-10.083	11.127	50	worse
Wave	300	200	29.231	0.712	3.470	26	NSD
Wave	3000	2000	24.881	0.922	1.262	29	better
Heart	3000	2039	2.555	-0.061	0.494	20	NSD
Oth. A	3000	2022	15.574	-5.349	0.943	20	worse
Oth. B	3000	6248	35.606	0.297	3.869	20	NSD
Hypo	5000	2438	0.744	0.051	0.095	20	better
Hyper	5000	2012	1.260	0.119	0.159	20	better
Binding	5000	2147	3.372	0.078	0.315	50	NSD
Replace	5000	2126	1.261	-0.103	0.237	50	worse
Euthy	5000	2218	0.874	-0.410	0.231	50	worse

The identical performance of the two methods in the case of Iris is not surprising; the pruned trees consist of only about seven nodes and the number of distinct values of the attributes is typically only about 30. The demon database delivered the expected result of

turning a negligible error rate into a large one, with a very high standard deviation.

Several of the domains were slightly improved by discretisation; this may be due to the fact that the discretised trees are typically smaller. Although all the trees were pruned in some cases further pruning may result in higher accuracy. Another possible explanation, more likely in the case of the thyroid domain, is that the thresholds can be more accurately determined from the full sample.

The differences on the Othello test (B) were insignificant; but when the test set is taken from the same game, predetermining the thresholds costs accuracy. This may be because there are very similar examples within a single game, which can be accurately classified by trees trees that use more complexity (additional tests on already tested attributes) than would be justified by what is true of the game in general. Ironically this domain does not really require further discretisation, ordered discrete attributes can be used directly, giving a large speedup with no loss of accuracy.

The binding domain was just within the null hypothesis with a lower bound for the confidence interval of -0.01. At a confidence level of 10% we can conclude that the discretised version was better. Two of the domains, replace and euthyroid, were significantly worse at the 5% level under discretisation, and the remaining two, hypo and hyper, were significantly better.

Some comparisons of learning time (including time to discretise) showed that in domains with only a few continuous attributes, learning time was approximately half in the discretised version. In the other domains (Othello, Wave & Heart), the speedup was by factors of more than 10.

Some comparisons with class-blind methods of discretisation largely gave the conclusion that their performance is inferior to D2 in these applications. The two methods tested were equal width discretisation (described in Section 3), and a simplification of maximum marginal entropy discretisation, which could be called "roughly equal population" discretisation; it simply entails sorting the n values (including repeats), and taking as the j thresholds the values of every j/nth value. We examined the performance of these methods with number of intervals arbitrarily chosen at 2, 7, and 24. For the wave domain, this gave much smaller trees at least as accurate as ID3's. (This is consistent with evidence from other sources that the trees grown by ID3 for the Wave domain are overly large, and should be pruned further. Binary discretisation is an extreme way of achieving this.) In almost all other domains the error rate for the class-

blind methods were grossly higher than for D2, with the exception of Othello, which can by explained by the following analysis. In the case of an attribute with only a few distinct values, n say, once the number of intervals reaches n, the discretised error rate (at least for "roughly equal population" discretisation) will become the same as the undiscretised version. In the case of Othello, the value of 24 is very close to $n = 29$. Learning time will of course be lowered by treating the attribute as ordered discrete.

A quick comparison between the two class-blind discretisation methods showed a strong difference only in the heart domain, where equal width discretisation performed very poorly, which is consistent with Wong & Chiu's preference.

6. Conclusion

We have shown that discretising continuous attributes can be achieved simply, offering very large savings in CPU time in domains where they abound. This change does not often result in a significant loss of accuracy (in fact it sometimes significantly improves accuracy).

Some work remains to be done to tune the basic algorithm for best performance. Evaluation should be extended to extremely large training sets. Here the the initial thresholding could be performed on a subset to cut down the computational cost of the thresholding.

Acknowledgments

Many people deserve thanks for providing data: Jeff Schlimmer for the Iris data, Ross Quinlan for the Thyroid data, John Oates and Ben Freedman of Royal Prince Alfred Hospital for the Heart data, and Paul Utgoff for the Othello data. Thanks are also due to Ross Quinlan for his advice and guidance. The anonymous referees of the European Working Session on Learning (EWSL-91) gave many very helpful comments and suggestions. This research was supported in part by a grant from the Australian Research Council.

References

Amarel, S. (1968). On the representation of problems of reasoning about action, In D. Michie (Ed.), *Machine Intelligence 3*, Edinburgh University Press.

Breiman, L., Friedman, J. H., Olshen, R. A., Stone C. J. (1984). *Classification and regression trees*. Belmont, CA: Wadsworth International Group.

Carter, C., & Catlett, J. (1987). Assessing credit card applications using machine

learning. *IEEE Expert, Fall 1987,* 71-79.

Kononenko, I., Bratko, I., & Roskar, E. (1984). Experiments in automatic learning of medical diagnostic rules, *Technical Report, Jozef Stefan Institute, Ljubljana.*

Michie, D. (1987). Current developments in expert systems. In J. R. Quinlan, (Ed.), *Applications of Expert Systems.* Maidenhead: Addison Wesley.

Michalski, R., Mozetic, T., Hong, J., Lavrac, N. (1986). The multi-purpose incremental learning system AQ15 and its testing application to three medical domains *Proceedings of AAAI-86,* Morgan Kaufmann.

Oates, J., Cellar, B., Bernstein, L., Bailey, B. P., Freedman, S. B. (1989). Real-time detection of ischemic ECG changes using quasi-orthogonal leads and artificial intelligence, *Proceedings, IEEE Computers in Cardiology Conference, 1989,* IEEE Computer Society.

Quinlan, J. R. (1979). Discovering rules by induction from large numbers of examples: a case study. In D. Michie (Ed.), *Expert systems in the micro-electronic age.* Edinburgh University Press.

Quinlan, J. R. (1983). Learning efficient classification procedures and their application to chess endgames (p. 469). In R. S. Michalski, J. R. Carbonell, T. M. Mitchell (Eds.), *Machine learning: an Artificial Intelligence approach* (pp. 463-82). Los Altos, CA: Morgan Kaufmann.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning, 1,1.*

Quinlan, J. R., Compton, P.J., Horn, K.A. & Lazarus, L. (1988). Inductive knowledge acquisition: a case study In J. Quinlan (Ed.), *Applications of expert systems,* Maidenhead: Addison-Wesley

Quinlan, J. R. (1987). Simplifying decision trees. *International Journal of Man-machine Studies, 27* (pp. 221-234).

Quinlan, J. R. (1987b). Decision trees as probabilistic classifiers, *Proceedings of the fourth international conference on machine learning,* (pp. 31-37) Morgan Kaufmann.

Quinlan, J. R. (1989). Unknown attribute values in induction *Proceedings of the sixth international conference on machine learning,* (pp. 164-168) Morgan Kaufmann.

Rendell, L. (1989). Comparing systems and analysing functions to improve constructive induction *Proceedings of the sixth international conference on machine learning* (pp. 461-464) Morgan Kaufmann.

Sejnowski, T. J., & Rosenberg, C. R., (1987). Parallel networks that learn to pronounce English text *Complex Systems 1.* (pp. 426-429).

Subramanian, D. (1989). Representational issues in machine learning *Proceedings of the sixth international conference on machine learning* (pp. 426-429) Morgan Kaufmann

Utgoff, P. & Heitman, P.S. (1988). Learning and generalizing move selection preferences *Proceedings of the AAAI symposium on computer game playing* pp. 36-40 (original not seen).

Utgoff, P. (1989). ID5: an incremental ID3 *Proceedings of the fifth international conference on machine learning* (pp. 107-120) Morgan Kaufmann.

Wilson, S. W. (1987). Classifier systems and the animat problem *Machine Learning, 2,4.*

Wirth, J., & Catlett, J. (1988). Costs and benefits of windowing in ID3 *Proceedings of the fifth international conference on machine learning* (pp. 87-99) Morgan Kaufmann.

Wong, A.K.C., Chiu, D.K.Y., (1987). Synthesizing statistical knowledge from incomplete mixed-mode data, IEEE Trans. Pattern Analysis and Machine Intelligence, November 1987, Vol PAMI-9, No. 6, pp. 796-805.