A Decision Support Tool for Tuning Parameters in a Machine Learning Algorithm

Margot Postema[*], Tim Menzies, and Xindong Wu

Department of Software Development
Monash University
900 Dandenong Road
Melbourne, VIC 3145, Australia

Email: {margot,timm,xindong}@insect.sd.monash.edu.au

Abstract

Many machine learning algorithms require parameter tuning in order to adapt them to the particulars of a training set. This tuning task can be an expert task in its own right. Based on our observations of an expert tuning the HCV (Version 2.0) rule induction algorithm, we have built a simple decision support system (DSS) to automate tuning for that algorithm. We found that HCV plus DSS produced equal or better rule sets than standard HCV or C4.5. A surprising result from this study is that a very simple approach to tuning worked very well. We speculate that other research on machine learner tuning have indulged in complex solutions before experimenting adequately with simpler alternatives.

Keywords: Decision support systems, machine learning, discretization, fine tuning

1 Introduction

The dream of machine learning is that we can automatically build specifications from data without requiring tedious and time consuming human involvement. To date, this dream has not been realised. A repeated experience in machine learning is that machine learning algorithms require parameter tuning in order to adapt them to the particulars of a training set. This tuning task can be an expert task in its own right. That is, machine learning has not yet removed the need for experts. Rather, it has changed the focus of the expertise required for building a specification.

In this study, we explore automating the machine learning tuning process. Based on our observations of the third author's use of a machine learning algorithm (HCV 2.0

*Thanks to Heinz Schmidt for suggestions and support with revisions.

Section 2) we developed a simple decision support system (DSS) for simplifying HCV parameter tuning (Section 3.2). When applied to datasets from the UCI repository, we find that HCV plus DSS produced equal or better rules sets than standard HCV or C4.5 in 9 of the 12 tested cases (Section 4). It is very simple to use our tuning DSS. So simple, in fact, that we believe that we could convert the DSS into a fully automatic expert system to handle the same task (Section 5). Hence, we speculate that parameter tuning of machine learning algorithms could be automated, thus realizing the dream of fully automated machine learning.

2 HCV (version 2.0): An Example Machine Learning Algorithm

The third author Wu, is the developer of the HCV rule induction algorithm [11]. This project began an experiment in trying to capture Wu's experience in tuning HCV. One particular area of Wu's expertise is in discretization of continuous variables. In the case of learning from data sets that contain continuous values, the values must be *discretized*; i.e. divided up into a finite set of contiguous ranges. Once divided up, then values for the continuous variables can be renamed; e.g. *high*, *low*. The essential aspect of discretization is to find the right places to set up interval borders.

In most machine learning algorithms discretization is an automatic process. In supervised discretization methods, such as the information gain based methods implemented in C4.5 [9] and HCV (Version 2.0) [11], the class information of examples in a training set is used. In unsupervised (or class-blind) discretization methods, such as equal-width discretization and equal-frequency discretization [2], training examples are grouped into intervals without taking into account the respective classes of the training examples. Discretization can be performed at induction time (such as in C4.5) or before induction takes place (see [3] and [7]). Among various discretization methods, the information gain based methods have been widely used and cited [3, 12]. C4.5 provides only binary discretization at induction time based on an information gain approach for real-valued attributes.

In HCV (Version 2.0), an information gain based method is the default discretization method for processing numerical attributes before induction takes place. This method is described below (Section 2.1.1) after a description of the HCV algorithm (Section 2.1).

2.1 The HCV (Version 2.0) Algorithm

The HCV algorithm [10] is a representative of the extension matrix based family of attribute-based induction algorithms, originating with J.R. Hong's AE1 [6]. 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 [11] 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 [11].

In addition to a set of discretization facilities, such as the information gain heuristic outlined in the next section, HCV (Version 2.0) permits the user to specify their own discretization 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 specific information. The tuning aid designed in Section 3.2 will start with this facility.

2.1.1 The Information Gain Heuristic for Discretization

When the examples in a 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 [8] to find a most informative border to split the value domain of the continuous attribute. Fayyad and Irani [4] have 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.

The information gain heuristic is adopted in HCV (Version 2.0) 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 [1] has proposed some criteria to stop the recursive splitting which have been adopted in HCV (Version 2.0):

- 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 in HCV (Version 2.0)), and
- Limit the number of intervals to be produced to a certain number (*e.g.* eight in HCV (Version 2.0)).

In C4.5 [9], 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 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 at each decision node.

3 A Fine Tuning Aid to HCV (Version 2.0)

3.1 Related Work

The tuning aid that we present in Section 3.2 is similar to *Adaptive Quantizers* discussed in Dougherty [3]. The Adaptive Quantizers method begins with a binary equal width partitioning of the continuous feature. Induction is performed and tested for predictive accuracy. The interval with the lowest predictive accuracy is split into two equal width partitions. The induction and evaluation processes are then repeated until some performance criteria is met. This method appears to overcome some of the limitations of unsupervised discretization, but comes with a high computational cost. This is due to numerous runs of the rule induction process. The method also assumes that a high level of accuracy can be achieved.

The minimal entropy heuristic [1, 3] uses class information entropy of candidate partitions to select the intervals.

The method we present uses some of the above mentioned algorithms slightly differently. Based on the HCV (Version 2.0) facility for user-specified intervals (Section 2.1), the tuning aid method for discretization is designed in Section 3.2.

3.2 Fine Tuning Design

We use an unsupervised learning method that firstly partitions the continuous attribute into ten equal width intervals. We then find three intervals based on the data, not the class distribution. These intervals are then incorporated into the induction algorithm. If the performance decreases (compared to the default) we don't do any further processing. Whilst the computational cost increases (Section 4.2), this is marginal when compared to the improvement of results. The hypothesis and tuning aid are presented in Sections 3.2.1 and 3.2.2 respectively.

3.2.1 Hypothesis

Initial experiments were devised based on the information available in the HCV *dictionary files*. These dictionary files list the names of the attributes and notes if they are discrete or continuous. From here a hypothesis was formed that the overall data distribution could be analysed to determine intervals in the *low, average* and *high* range. This is achieved by plotting a histogram of the data in the training files into ten equal intervals. These intervals were then analysed to find the following three intervals:

- *Low* which contained approximately 25% of the data
- *Average* which contained approximately 50% of the data
- *High* which contained the remaining data

Since it is not possible to use this method when a large percentage (say 70%) of the data lies at either extremes of the range of values, a second method is introduced which includes overlapping class intervals. For example, if class 1 has values ranging 0-100 and class 2 has values ranging 85-150, the intervals used are 0-100 and 85-150. This method had limited use and success.

3.2.2 Tuning Aid

1. Run the HCV (Version 2.0) software on a database to obtain baseline results. The database should consist of at least one continuous attribute.

2. Select a continuous attribute in the database.

3. Check the overall data distribution on the selected attribute.

4. Divide the continuous attribute into the following three intervals:

    - low (approximately 25% of the data)
    - average (approximately 50% of the data)
    - high (approximately 25% of the data)

    If the above fails (eg. 70% of data lies at one interval) attempt to determine overlapping class intervals (eg. 0-100, 85-150, as mentioned in Section 3.2.1).

5. If intervals were found in step 4, run the HCV (Version 2.0) software with these intervals and obtain results, else go to step 7

6. If the results are improved or no worse than the baseline ones, keep them, else discard these intervals.

7. If no more continuous attributes, STOP.

8. Select another continuous attribute.

9. Go to step 3.

The second method discussed in Section 3.2.1 and Step 4 above deals with domains with some degree of fuzziness. For example, we could classify someone who is 170cm in height as 'Normal Height' or 'Tall Height' depending on our outlook. Instead of discretizing the domain into fixed intervals, some sort of curve could be applied to give a fuzzy border. Thus a value could belong to more than one interval at the same time. Results can be classified in terms of degrees of membership in each interval. Discretizing a continuous domain in this way is termed *fuzzy matching* [13]. Alternatively, the number of intervals and their values could also be modified, which is the principle used in fuzzy expert systems. This above method combines probabilistic analysis (refer to Step 4) and fuzzy logic.

The method based on the information in the dictionary file and the data in the training files can be related to the probability surveys between domain experts when designing a fuzzy expert system [5]. A summary of the tuning aid method can be included into the machine learner as shown by Figure 1, where the user can specify the intervals from the training examples. Following the heavy arrowed lines, we can see:

1. Training and test data are fed into the machine learner as a baseline.
2. The user selects a continuous attribute from the database.
3. The user checks the data distribution and
4. specifies intervals.
5. Feed these intervals into the machine learner.
6. The deduction of induction results on test data are then compared to determine which method gave the best results.
7. A continuous variable is either discretized by the user, or the default machine learner intervals are selected. The dotted lines indicate the user decision.

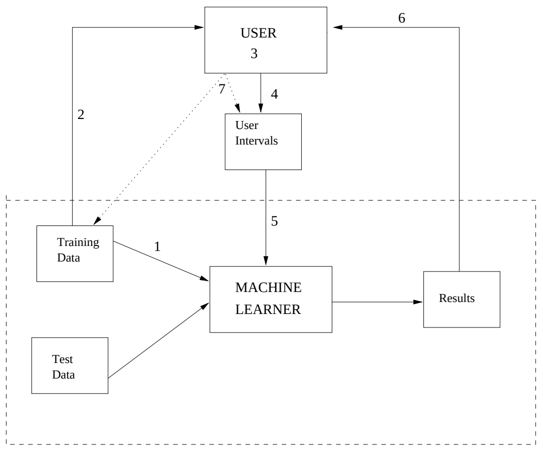

Figure 1: The Machine Learning Cycle

4 Experiments

The experiments were performed with data sets from the University of California at Irvine Repository of Machine Learning Databases and Domain Theories. The inputs for the machine learners include the *dictionary, training* and *testing* files, which describe the attributes, and include training and testing examples respectively.

4.1 Baseline

The experimental results can be compared to those obtained by Wu et al. [14]. The best results in terms of predictive accuracy for each problem is given in *bold.*

4.2 Results

Table 1: Success Analysis of Results

Domain	*Success Analysis*		
	$\odot$	$\ominus$	$\oplus$
Wine	-	4	5
Bupa	-	4	2
Labor Neg	-	4	2
Swiss 5	-	3	1
Cleveland 2	-	-	1
Cleveland 5	1	-	-
Va 5	-	-	1
Va 2	-	1	-
Crx	-	-	1
Imports 85	4	8	2
Pima	1	5	1
Glass (without ID)	-	6	-

Legend

$\odot$ indicates the same accuracy (the intervals were included in the subsequent experiments).
$\ominus$ indicates decreased result (subsequently discarded).
$\oplus$ indicates improved result.

Table 1 analyses the success of using the data distribution method defined in Section 3.2 to specify the intervals in continuous domains. The experimental results are summarised for all the continuous attributes of each data set. One interesting observation is that the probabilistic method is actually a search for attributes that can be discretized usefully. In the twelve data sets, 16 of the 55 continuous attributes experimented with give improved results, 34 give worse results and 5 make no difference.

The induction algorithm is run $N + 1$ times where N represents the number of continuous attributes, thereby increasing the computational cost by N times.

The experiments including the probabilistic method are recorded as HCV (Tuned) in Table 2. The overall success in terms of accuracy is high if we consider that results from HCV (Tuned) have the highest accuracy on 7 out of the 12 data sets tested, and matched C4.5 in another two.

Table 2: Summary and Comparison of Results

	Accuracy (%)		
Domain	**C4.5**	**HCV (Ver2.0)**	**HCV (Tuned)**
Bupa	**73.7**	57.6	65.3
Cleveland 5	47.3	**56.0**	**56.0**
Cleveland 2	68.4	78.0	**82.4**
Crx	79.5	82.5	**85.0**
Imports 85	64.4	62.7	**66.1**
Labor Neg	**82.4**	76.5	**82.4**
Pima	**75.1**	73.9	**75.1**
Swiss 5	31.2	28.1	**34.4**
Va 5	26.8	26.8	**32.4**
Va 2	66.7	**78.9**	70.4
Wine	90.4	90.4	**98.1**
Glass (without ID)	64.6	**72.3**	69.2

5 Conclusion

We have described the construction of a simple decision support system for tuning the discritization process of a rule induction algorithm. Users of this tool must view some histograms and propose border values for creating discrete intervals. Experiments with this tool and HCV (Version 2.0) show that learning + tuning out-performs (in the majority of cases) learning alone.

This process is very quick and simple. The process is so simple that we could fully automate the process thus turning a decision support system requiring human interaction into a true expert system that required no expert input.

One interesting result from the above study is that a very simple approach to tuning worked very well. We wonder if some of the discussions on learner tuning have indulged in complex solutions before experimenting adequately with simpler alternatives.

Future work in this area includes:

- Exploring the use of such simple tuning tools to other learners like C4.5
- Exploring the utility of having more than 3 discrete division (e.g. 5).

References

[1] J. Catlett. On Changing Continuous Attributes into Ordered Discrete Attributes. *Proceedings of the 1991 European Working Session on Learning*, 1991.

[2] D. Chiu, A. Wong and B. Cheung. Information Discovery through Hierarchical Maximum Entropy Discretization and Synthesis. In Piatetsky-Shapiro, G., and W.J. Frowley, editors, *Knowledge Discovery in Databases*. MIT Press, 1991.

[3] J. Dougherty, R. Kohavi and M. Sahami. Supervised and Unsupervised Discretization of Continuous Features. *Proceedings of the 12th International Conference on Machine Learning*, 194–202, 1995.

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

[5] L.O. Hall and A. Kandel. The Evolution of Expert Systems. In F. Aminzadeh and M. Jamshidi, editors, *Soft Computing: Fuzzy Logic, Neural Networks, and Distributed Artificial Intelligence.* Prentice Hall, New Jersey USA, 1994.

[6] J.R. Hong. AE1: An Extension Matrix Approximate Method for the General Covering Problem. *International Journal of Computer and Information Sciences*, 4(6): 421–437, 1985.

[7] B. Pfahringer. Compression-Based Discretization of Continuous Attributes, *Proceedings of the 12th International Conference on Machine Learning*, 456–463, 1995.

[8] J.R. Quinlan. Induction of Decision Trees. *Machine Learning, 1.* 81–106, 1986.

[9] J.R. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann Publishers, 1993.

[10] X. Wu. The HCV Induction Algorithm. In S.C. Kwasny and J.F. Buck, editors, *Proceedings of the 21st ACM Computer Science Conference*,168–175. ACM Press, USA, 1993.

[11] X. Wu. *Knowledge Acquisition from Databases.* Ablex, USA, 1995.

[12] X. Wu. A Bayesian Discretizer for Real-Valued Attributes. *The Computer Journal.* **39**(1996).

[13] X. Wu and P. Måhlén. Fuzzy Interpretation of Induction Results. *Proc. of the 1995 International Conference on Knowledge Discovery and Data Mining (KDD-95)*, Montreal, Canada, August 20-21, 1995, 325–330.

[14] X. Wu, J. Krisár, and P. Måhlén. Noise Handling with Extension Matrices. In *International Journal of Artificial Intelligence Tools*, **5**(1996), 81-97.