A Family of Efficient Rule Generators

Huan Liu

Department of Information Systems and Computer Science

National University of Singapore

Kent Ridge, Singapore 119260

liuh@iscs.nus.sg

Tel: (+65)-772-6563; Fax: (+65) 779-4580

Abstract

This paper describes a family of rule generators that can be used to extract classification rules in various applications. It includes versions that can handle noise in data, that can produce perfect rules, that can induce order independent or dependent rules based on a core algorithm. Evaluation measures are given to compare different rule generators in the family and with some known methods. Empirical results and applications of these rule generators in the family show that they are efficient, compact and accurate in various situations. The systems are available for trial use upon request.

Key Words: Classification, Rule Compactness and Comprehensibility, Noise Handling

I. Introduction

Classification rules are sought in many areas from automatic knowledge acquisition [15, 16] to data mining [1, 22], neural network rule extraction [19, 6, 17]. This is because classification rules possess some attractive features. They are explicit, understandable and verifiable by domain experts, and can be modified, extended and passed on as modular knowledge.

Although many different models of induction, such as decision trees, neural networks, and linear discriminants, have been used for classification, they share a common goal: prediction accuracy. Model complexity and prediction accuracy are highly related. Learning from sample data can be described as estimating a model's parameters from a view point of Statistics. There are strong theoretical reasons for developing learning methods that cover patterns in the most efficient and compact manner [20, 21]. This paper describes our efforts toward finding simple classification rules with high prediction accuracy.

This work stems from practical needs in knowledge acquisition, neural network rule extraction, as well as data mining. To cater for different needs, variations of rule generators are implemented centering around a base algorithm. This work can find its relevance in decision tree rule induction [15], but is more related to single-best-rule covering [13, 22] that expands only one rule at a time and add conditions one by one until the rule set covers all the training patterns. The distinction of this work from the related work mainly lies in how to efficiently add these conditions in a rule plus some special considerations for particular applications. Details will be given later. Another line of related work is of AQ11 and its variations [3]. However, AQ11 [18] is a candiate-elimnation algorithm that requires a generalization hierarchy in the search of learned concepts. This is an advantage when the domain knowledge is available. Here we assume no domain knowledge.

In the following section, we describe the base algorithm. In Section 3, we explain how different family members are constructed by modifying the base algorithm. In Section 4, we introduce two evaluation measures for prediction accuracy and compactness. In Section 5, we account for the experiment method, and present empirical results for comparative study with a discussion on issues as efficiency, etc. In the last section, we conclude this project and propose some future work.

II. Base Algorithm

The aim of this work is to search for simple rules with high prediction accuracy. The basic idea of the proposed algorithm is simple: using first order[1] information in the data to determine shortest sufficient conditions in a pattern (i.e., the rule under consideration) that can differentiate the pattern from patterns belonging to other classes. The sole use of first order information avoids the combinatorial complexity in computation, although it is well known that using higher order information may provide better results.

The pseudo code of the base algorithm for rule generation is given in Figure 1. The technical details are given to the extent that a re-implementation of the algorithm can be done without much difficulty. The variations of the rule generator are produced by modifying, adding/deleting some key components of the base algorithm. Each variation is discussed in the next section.

The base algorithm is composed of three major functions:

[1]The degree of order is determined by the number of attributes involved in search. If one attribute is used, the search only makes use of first order information.

1. Rule generation – this function iteratively generates shortest rules[2] and removes/marks the patterns covered by each rule until all patterns are covered by the rules.

2. Rule clustering – rules are clustered in terms of their class labels; and

3. Rule pruning – redundant or more specific rules in each cluster are removed.

A default rule should be chosen to accommodate possible unclassifiable patterns. If rules are clustered, the choice of the default rule is based on clusters of rules.

In addition to the three functions, one of the key concepts of the algorithm is *generating the <u>shortest</u> rule* R_i at each iteration i. The way of finding the shortest rule determines the efficiency of the algorithm. Our method is straightforward, and makes use of first order information only. It finds the shortest sufficient conditions for each rule of a class. x/y means choosing either x or y. The pseudo-code of this algorithm is given in Figure 2.

III. Family of Rule Generators

Centering around the base algorithm, we describe four rule generators in turn.

A. Order dependent version

By being order dependent we mean that the order of the rules cannot be changed. In other words, the first rule is always tested first. An order dependent (OD) rule generator produces OD rules by removing all patterns covered by each new rule until the data set is empty. In the base algorithm, if *empty* and *remove* remain in the

[2] A shortest rule is a rule with a minimum number of conditions.

Base Algorithm

```
Rule Generation:
   i=0;
   while (Data is NOT  empty/marked) {
      generate  shortest R_i to
         cover the current pattern and
         differentiate it from patterns in other categories;
      remove/mark all the patterns covered by R_i;
      i++;
   }
Rule Clustering:
   cluster rules according to their class labels;
Rule Pruning:
   eliminate redundant rules;
determine a default rule.
```

Figure 1: The pseudo-code of the Base Algorithm.

Generating the Shortest Rule R_j

```
for each attribute value v of the current pattern p of class c_i,
    count v's occurrence in the patterns
       belonging to other classes (≠ c_i),
order the attributes according to their value occurrences
   in an increasing sequence A_freq[ ] and its sister
   sequence A_order[ ] storing the indexes of attributes,
if A_freq[1] = 0
   (value v alone can differentiate pattern p from others)
   obtain R_j as: if A_k = v then c_i,
        k is determined by A_order[1];    ---- (1)
else
   find A_freq[1 - n] that can uniquely represent pattern p
      by increasing n from 2 to the maximum number of attributes,
   obtain R_j as in ---- (1) by ANDing all the chosen attributes.
```

Figure 2: The pseudo-code of finding the shortest rules.

while loop, an OD rule generator is obtained. Because the rules are ordered, Rule Clustering and Rule Pruning components cannot apply and should be deleted from the base algorithm thereof.

B. Order independent version

By being order independent we mean that the firing order of the rules can be changed without affecting the result of classifiction. An order independent (OI) rule generator forms OI rules by keeping all the patterns for generating each new rule. In effect, an OI rule generator results from keeping *marked* and *mark* in the while loop of the base algorithm. To *mark* a pattern is to register the pattern as having been considered. When all patterns are considered (*marked*) for rule generation, the algorithm stops.

C. Binary version - perfect rules

By and large, rule generators [14, 2, 15] and other versions of this work focus on generalization, avoiding overfit of the training data. This version is to generate perfect rules as it is required in neural network rule extraction applications [17, 10]. By perfect rules we mean that rules can cover the patterns with a zero percent error rate. In some cases such as neural network rule extraction, the training data for rule induction is noise free and all patterns are equally important. The two features distinguish this version from others are: 1. that only binary data is used, and 2. that patterns are grouped in terms of class labels before learning begins. Apart from these two features, this version per se is an order independent rule generator with a special treatment for binary data. Since this version is specifically designed for neural network rule extraction based on the base algorithm, no further discussion will be given.

D. Noise handling version

In choosing which pattern to start with, the base algorithm is data-order sensitive in the sense that the rule set could be different if the order of patterns is changed, or patterns are shuffled. There would be no problem if no patterns are corrupted. In presence of noise (or corrupted patterns), however, it is important to start with a noise free pattern. Therefore, in order to handle noise, some additional information is required. One piece of information is the frequency of each distinct pattern in the training data. Under the assumption that there are only a small number of corrupted patterns, if we begin with a pattern that occurs most frequently, it is then remotely likely that it is a corrupted pattern. In order to handle noise, the base algorithm is modified by adding the following at the very beginning of the algorithm.

```
sort-on-freq(Data-without-Duplicates);
```

The frequency of each pattern can be obtained by counting each distinct pattern's occurrence and removing duplicates. The output of Chi2 [9], which is a program that does discretization and feature selection for numeric data, also provides such information. More on handling noise in various cases can be found in [8].

IV. Evaluation Measures

The evaluation of rules is performed in two aspects. One is the estimation of error rates [11]. In a sense, it measures how well the rules generalize from data since usually the data is split into two sets, one for training, the other for testing. The lower the error rate for the testing set is, the better the rules generalize from the data. Rules that do not generalize are of minimum use. However, if there are two sets of rules and their estimated error rates are similar, the comparison of the two should be done in terms of the compactness of a rule set. As was mentioned in Introduction, a simpler set is preferred.

- **Error rates**. The training data is used to learn rules. What we would like to know is the proportion of errors made by this set of rules, and classifying new observations (the testing data in experiments) without the benefit of knowing the true classifications. The true classifications of a testing data are known but are not told to the rule generator. The most common evaluation method is called "one-shot" train-and-test which applies rules to the data, counts the errors made, and calculates the error rate as the number of errors divided by the number of data. When the testing data is not available, this evaluation is done on the training data. As we know, it is very biased. The more sophisticated methods are cross-validation (m-fold, leave-one-out, etc.) and bootstrap [11].

- **Compactness measure**. An ideal rule set has a minimum number of rules and each rule is as short as possible. In practice, it is quite often that a rule set contains fewer rules but some of which have more conditions. Therefore, the concept of compactness is multi-dimensional. The dimensions that need be considered are: 1. the number of rules in a set; 2. the number of conditions in a rule; and 3. the number of patterns covered and correctly classified by a rule. The following compactness measure was proposed in [22]:

$$E = \frac{1}{r}\sum_{i=1}^{r}\frac{t_i}{n}\frac{1}{c_i}, \tag{1}$$

  where r is the cardinality of the rule set, n the number of patterns, t_i the number of patterns correctly classified by rule i, $1 \leq i \leq r$, and c_i the number of conditions in rule i.

One problem with this compactness formula is that it cannot handle the default rule, since $c = 0$ for the default rule, which is often very useful to accommodate those patterns that cannot be covered by any other rules. An adjustment is to let its c be 1 for the default rule. Another problem will be discussed in the end of Section 5 about the use of the formula for comparison.

V. Experiments

Equipped with the above two evaluation measures, we can conduct some experiments to empirically compare the four versions of rule generators, and with other known methods [13, 22].

A. Method

One data set - CAR will be used to show the differences between the versions of our rule generator, and to compare with the results reported in [22] since they have done some comparison with other methods such as ID3 [14] and the one by Han et al [7].

Then, we show the results for another two data sets: Golf-Playing [13] and Iris [4]. The authors of [13, 22] did not provide testing data. Only the Iris data is divided evenly into two sets (75 patterns each) for training and testing. Datasets CAR and Golf-Playing are given in Tables 1 and 2. Iris data can be obtained from University of California, Irvine [12].

In presence of numeric attributes (e.g., in Golf-Playing and Iris data), Chi2 [9] is applied to discretizing these attributes. The resulting data contains discrete values only and is guaranteed that it can keep the original discriminating power of the data, shown in Tables 3 and 4.

displace	fuelcap	mass	speed	cyl	cost
3	3	2	2	6	3
3	1	3	3	6	3
2	2	1	3	6	2
1	1	1	1	6	1
3	2	2	2	6	3
3	2	1	2	6	3
1	1	1	2	6	1
1	2	1	1	4	1
2	1	2	2	6	2
2	3	2	3	6	3
2	3	1	3	6	3
1	3	3	2	4	3
1	3	1	2	4	1
2	1	3	2	4	3
2	2	2	2	4	2
2	3	2	2	4	2
1	2	2	3	4	2
2	2	3	1	4	3

Table 1: CAR dataset. Displace: 1 - small, 2 - medium, 3 - large; Fuelcap: 1 - low, 2 - medium, 3 - heavy; Mass: 1 - light, 2 - medium, 3 - heavy; Speed: 1 - slow, 2 - medium, 3 - fast; Cost: 1 - cheap, 2 - medium, 3 - expensive.

Outlook	Temp.	Humid.	Wind	Decision
1	21	96	1	0
1	17	70	1	0
2	21	90	1	1
2	29	78	2	1
1	23	80	2	1
2	16	66	1	1
3	23	70	1	1
3	27	90	1	0
3	30	85	2	0
2	27	75	2	1
1	18	80	2	1
1	20	96	2	1
3	21	95	2	0
3	19	70	2	1

Table 2: Golf-Playing dataset. Outlook: 1 - rainy, 2 - overcast, 3 - sunny; Wind: 1 - strong, 2 - light; Decision: 0 - don't play, 1 - play.

B. Detailed results

In the following comparative studies, three data sets (CAR, Golf-Playing, Iris) are used respectively. Noise handling is shown in Comparison 3 since the first two data sets do not have any noise. Some notations are: '0' means a wild card, and '-' means *not*. The noise in Iris data was purposefully introduced in data discretization.

Comparison 1 (CAR): Given below are the rules and relevant information for calculating the compactness of each rule set.

```
--- A complete set of rules (order independent) ---

Rule0: 0 0 3 0 0 -> 3

Hit: 4 and Miss: 0


Rule1: 3 0 0 0 0 -> 3
```

```
Hit: 3 and Miss: 0


Rule6: 0 3 0 3 0 -> 3

Hit: 2 and Miss: 0


Rule7: 1 0 1 0 0 -> 1

Hit: 4 and Miss: 0


Default: 2

Hit: 5 and Miss: 0


--- Testing rules for training data: ---

Total #Item: 18; Hit: 18; Miss: 0; Acc: 1.00


E = 15/18/5 = 0.166 according to formula (1).

--- A complete set of rules (order dependent) ---

Rule0: 0 0 3 0 0 -> 3

Hit: 4 and Miss: 0


Rule1: 3 0 0 0 0 -> 3

Hit: 3 and Miss: 0


Rule2: 2 2 0 0 0 -> 2

Hit: 2 and Miss: 0
```

```
Rule3: 0 0 0 1 0 -> 1

Hit: 2 and Miss: 0


Rule4: 1 1 0 0 0 -> 1

Hit: 1 and Miss: 0


Rule5: 0 1 0 0 0 -> 2

Hit: 1 and Miss: 0


Rule6: 0 0 0 0 6 -> 3

Hit: 2 and Miss: 0


Rule7: 0 0 1 0 0 -> 1

Hit: 1 and Miss: 0


Default: 2

Hit: 2 and Miss: 0


--- Testing rules for training data: ---

Total #Item: 18; Hit: 18; Miss: 0; Acc: 1.00


E = 16.5/18/8 = 0.115 according to formula (1).
```

The compactness values for the three systems reported in [22] are 0.12, 0.062, and

Outlook	Temp.	Humid.	Wind	Decision	Freq.
1	0	2	1	0	1
1	0	1	1	0	1
3	0	2	1	0	1
3	0	2	2	0	2
2	0	1	1	1	1
3	0	1	2	1	1
3	0	1	1	1	1
2	0	1	2	1	2
1	0	1	2	1	2
2	0	2	1	1	1
1	0	2	2	1	1

Table 3: Discretized Golf-Playing data.

0.014 for LCR of [22], ID3 and Han et al's algorithm, respectively. It seems obvious that the order independent (OI) rule set is more compact than the order dependent (OD) one since the former has much fewer rules. When the latter comparing to the rule set generated by LCR, although LCR generates six rules only, the difference between the two (0.115 vs. 0.12) is rather insignificant. This may be due to the OD rules having fewer conditions. ID3 generates nine rules, four of which contain three conditions each.

Comparison 2 (Golf-Playing): The discretized data is shown in Table 3. Here we continue to see the difference between the rule sets generated by the OI and OD versions. A rule (Rule 6) is pruned in the OI version since there is another rule (Rule 3) that is more general.

```
--- A complete set of rules (order independent) ---
Rule1: 3 0 2 0 -> 0
Hit: 3 and Miss: 0
```

```
Rule3: 1 0 0 1 -> 0

Hit: 2 and Miss: 0


*Rule6: 1 0 2 1 -> 0 (pruned)


Default: 1

Hit: 9 and Miss: 0


--- Testing rules: ---

Total Item: 14; Hit: 14; Miss: 0; Acc: 1.00

--- A complete set of rules (order dependent) ---

Rule1: 3 0 2 0 -> 0

Hit: 3 and Miss: 0


Rule4: 1 0 0 -2 -> 0

Hit: 2 and Miss: 0


Default: 1

Hit: 9 and Miss: 0


--- Testing rules: ---

Total Item: 14; Hit: 14; Miss: 0; Acc: 1.00
```

The two sets generated by the OI and OD versions are not exactly the same but their compactness values are because both sets of rules have the same number of

S-len	S-wid	P-len	P-wid	Class	Freq
0	0	1	1	0	25
0	0	2	2	1	24
0	0	2	3	1	1
0	0	3	3	2	21
0	0	2	3	2	3
0	0	3	2	2	1

(a) training data

S-len	S-wid	P-len	P-wid	Class	Freq
0	0	1	1	0	25
0	0	2	2	1	23
0	0	3	3	1	1
0	0	3	2	1	1
0	0	3	3	2	19
0	0	3	2	2	3
0	0	2	3	2	3

(b) testing data

Table 4: Discretized Iris data obtained by Chi2. S - sepal, P - petal, len - length, wid - width, freq - frequency. The training data is shown in table (a), and the testing data is shown in table (b).

rules, number of conditions, as well as number of Hits and Misses. In this case, we prefer the OI rule set since it can be understood without concern of ordering. The results reported in [13] are: RULES-2 produced 8 and 14 rules for option 1 and 2. Under the normal circumestances, we prefer a rule set with fewer and shorter rules.

Comparison 3 (Iris): The discretized training and testing data sets are shown in Table 4 (a) and (b). The noise handling version is applied to get the results as follows. Both OI and OD versions produce the same rule set.

```
--- A complete set of rules ---

Rule0: 0 0 1 0 -> 0
```

```
Hit: 25 and Miss: 0


Rule1: 0 0 2 2 -> 1

Hit: 24 and Miss: 0


Default: 2

Hit: 25 and Miss: 1


--- Testing rules for training data: ---


Total #Item: 75; Hit: 74; Miss: 1; Acc: 0.99


--- Testing rules for testing data: ---


Total #Item: 75; Hit: 73; Miss: 2; Acc: 0.97
```

Translating the discretized data back to their original meanings, we obtain the following rules:

Rule 1: If Petal-length ≤ 1.9 then *Iris setosa*

Rule 2: If Petal-length ≤ 4.9 and Petal-width ≤ 1.6 then *Iris versicolor*

Default Rule: *Iris virginica.*

For reference, the rule set (DT Rules) generated by C4.5rules [15] (which is based on a decision tree method but generates more concise rules than the tree itself) is included here:

Rule 1: If Petal-length ≤ 1.9 then *Iris setosa*

Rule 2: If Petal-length > 1.9 and Petal-width ≤ 1.6 then *Iris versicolor*

Rule 3: If Petal-width > 1.6 then *Iris virginica*

Default Rule: *Iris setosa.*

C. Results on real-world datasets

In the above, the detailed results are reported on the several datasets used in the cited references. All these datasets are relatively small. Here, two relatively large datasets are chosen for further experiments from the machine learning databases at University of California, Irvine [12]. They are:

- **Mushroom** The training and testing datasets contains 7124 and 1000 instances respectively. The 1000 instances in the testing set are randomly selected. The rest are used for training. The data has 22 discrete attributes. Each attribute can have 2 to 10 values.

- **Wisconsin Breast Cancer** The training and testing datasets contains 350 and 349 instances respectively. 350 instances are randomly selected for training, the other half is for testing. There are 9 discrete attributes. Each attribute has 10 values.

The results of running the order-dependent (OD) and order-independent (OI) rule generators are summarized below, following the style in [5], to provide the accuracy (Acc) and number of rules (# Rules). The accuracy for the training datasets is 100%

in all cases. The reported accuracy below is for testing data only.

Dataset	OD Acc.	OI Acc.	OD # Rules	OI # Rules
Mushroom	100%	99%	12	11
Breast Cancer	93%	92%	20	12

Due to the relatively large number of rules, the rules are omitted here and available upon request. Some extra information about rules may be helpful to the readers who are familiar with the two datasets. For the mushroom dataset, the rules only contain up to four attributes which are attributes 4, 5, 12 and 22; and for the breast cancer dataset, the rules only need a maximum number of three attributes which are attributes 1, 2 and 6. The above reported accuracy rates are comparable to the best ones reported in the literature.

D. Discussion

- *Efficiency issue.* Given in [13], the computational complexity of RULES-2 in its worst case is:

$$N_{RULES-2} = O(n^2 \sum_{i=1}^{m} \frac{m!}{(m-i)!i!} i) \geq O(n^2 2^m),$$

where n is the number of patterns, m (> 1) the number of attributes for the given problem.

Suppose the average number of distinct values in each attribute is h, LCR's computational complexity given in [22] in its worst case is:

$$N_{LCR} = O(\sum_{i=1}^{m} \frac{h^i m!}{(m-i)!i!} n * i).$$

In addition, the LCR algorithm requires a lot of memory to store tables for calculating Semantic Association Degrees.

The computational complexity of the base algorithm in its worst case is:

$$N = O(n^2 m^2).$$

It is clear that N is smaller than $N_{RULES-2}$. A comparison of N to N_{LCR} is not straightforward due to N_{LCR}'s complex representation as well as its reliance on the massive memory usage. Consider only the complexity of its last term $i = m$ (the smallest in the factorial), its complexity is $O(n_{max}nm)$. Hence, it is conjectured that N and N_{LCR} are similar. For a reference of the scale of these complexities, the worst case computational complexity of a decision tree method is n^2mC, where C is the cost to calculate a split function (e.g., information gain in ID3). All these worst case computational complexities are approximate. For example, N is rough since at any time only k (approximated by n divided by the number of classes) patterns are checked, and on average, $m \simeq 2$.

- *Order dependent vs. order independent.* The two versions have their pros and cons. It is natural to have OI rules since all matched rules will have a chance to get fired. However, a conflict resolution mechanism is normally required to choose which is the best rule to be fired. The disadvantage of OD rules is that the ordering hinders our understanding of the rules. As was suggested in Comparison 1, the OD rules tend to have fewer conditions.

- *Compactness measure.* From the experiments, especially Comparison 1, it is clear that the measure favors rules with a smaller number of conditions. If each rule has only one condition, even if the number of rules is equal to the number of patterns, the measure would become $\frac{1}{r}$. In the example of CAR, it is 1/18 = 0.055 which is still more compact, according to formula (1), than the rules produced by Han et al's algorithm. This does not seem reasonable.

VI. Conclusion and Further Work

This paper introduces a basic rule generation algorithm and its variations for different applications. In essence, the algorithm exploits the first order information in the data and finds shortest sufficient conditions for a rule of a class that can differentiate it from patterns of other classes. Two evaluation measures (accuracy and compactness) are used in comparative studies. Although the rules produced by the versions of our rule generator are different, they are compact and accurate. Noise handling of the algorithm is effective. It is best used in conjunction with Chi2 [9] since Chi2 provides the frequency information. Rule pruning removes redundant rules.

Four rule generators can be used in different circumstances as their names suggest. In summary, if the induced rules are not employed in sequence, the order dependent rule generator should not be considered, although it may produce shorter rules than other versions. If the frequency information is not available, the noise handling rule generator should not be considered. If the comprehensibility of rules is the top concern, the order independent rule generator should be used. In case of binary data, the binary version is strongly suggested since it takes into account the special features of binary data.

The future work includes: 1. searching for a better compactness formula to deal with the factors that are not well expressed in the current formula (1); 2. handling noise data without the frequency information, as this information may not always be available; and 3. improving the base algorithm by making the conditions of a rule not only sufficient but also necessary while keeping the scale of the current computational complexity.

References

[1] R. Agrawal, T. Imielinski, and A. Swami. Database mining: A performance perspective. *IEEE Trans. on Knowledge and Data Engineering*, 5(6):914–925, December 1993.

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

[3] P.R. Cohen and E. A. Feigenbaum. *The Handbook of Artificial Intelligence III*. William Kaufmann Inc., 1982.

[4] R.A. Fisher. The use of multiple measurements in taxonomic problems. *Ann. Eugenics*, 7(2):179–188, 1936.

[5] P.W. Frey and D.J. Slate. Letter recognition using holland-style adaptive classifiers. *Machine Learning*, 6(2):161–182, 1991.

[6] L. Fu. *Neural Networks in Computer Intelligence*. McGraw-Hill, 1994.

[7] J. Han, Y. Cai, and H. Cercone. Data-driven discovery of quantitative rules in relational databases. *IEEE Trans. on Knowledge and Data Engineering*, 5(1):15–28, 1993.

[8] H. Liu. Efficient rule induction from noisy data. *Expert Systems with Applications: An International Journal*, 10(2):275–280, 1996.

[9] H. Liu and R. Setiono. Chi2: Feature selection and discretization of numeric attributes. In *Proceedings of the 7th IEEE International Conference on Tools with Artificial Intelligence*, 1995.

[10] H. Lu, R. Setiono, and H. Liu. Neurorule: A connectionist approach to data mining. In *Proceedings of VLDB'95*, pages 478–489. Elsevier Science Ltd, 1995.

[11] D. Michie, D.J. Spiegelhalter, and C.C. Taylor. *Machine Learning, Neural and Statistical Classification*. Ellis Horwood Series in Artificial Intelligence, 1994.

[12] P.M. Murphy and D.W. Aha. UCI repository of machine learning databases. FTP from ics.uci.edu in the directory pub/machine-learning-databases, 1994.

[13] D.T. Pham and M.S. Aksoy. An algorithm for automatic rule induction. *Artificial Intelligence in Engineering*, 8, 1994.

[14] J.R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

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

[16] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1995.

[17] R. Setiono and H. Liu. Understanding neural networks via rule extraction. In *IJCAI-95, Proceedings International Joint Conference on Artificial Intelligence*, 1995.

[18] C.J. Thornton. *Techniques of Computational Learning: an introduction*. Chapman and Hall, 1992.

[19] G.G. Towell and J.W. Shavlik. Extracting refined rules from knowledge-based neural networks. *Machine Learning*, 13(1):71–101, 1993.

[20] C. Wallace and P. Freeman. Estimation and inference by compact function. *J. Royal Statistical Soc. Series B.*, 49B(3):240–265, 1987.

[21] S.M. Weiss and N. Indurkhya. Optimized rule induction. *IEEE Expert*, 8(6):61–69, December 1991.

[22] S-J Yen and A.L.P. Chen. An efficient algorithm for deriving compact rules from databases. In *Proceedings of the Fourth International Conference on Database Systems for Advanced Applications*, 1995.