

Honours Year Project Report

Rule Generation from Discrete Data

By

Gong Tianxia

Department of Computer Science

School of Computing

National University of Singapore

2005/2006

Honours Year Project Report

Rule Generation from Discrete Data

By

Gong Tianxia

Department of Computer Science

School of Computing

National University of Singapore

2005/2006

Project No: H001090

Advisor: Prof Rudy Setiono

Deliverables:

Report: 1 Volume

Program: 1 CD

Abstract

Greedy Rule Generation (GRG) is a rule generation algorithm in machine learning. It learns the examples and generates an ordered rule list. It differs from other rule generation algorithms mainly in the division of the examples space into subspaces and the rule merging process. We added an upper limit to the subspaces size and thus guaranteed small time complexity and rule list complexity. Chi2 discretization and information gain measures are used for feature discretization and selection. We tested our approach intensively on 14 data sets in the experiment. GRG has shown competitive performance in comparison with M5' and N2C2S and better performance over rule generation algorithms as AQ and CN2.

Subject Descriptors:

- I.2.6 Learning
- I.1.2 Algorithms
- G.3 Probability and Statistics

Keywords:

Greedy rule generation, Chi2 discretization, ten fold cross-validation

Implementation Software and Hardware:

- Software: Eclipse, WEKA, MS Windows XP
- Hardware: Intel Pentium 3.0GHz CPU, Kingston 512 RAM

Acknowledgment

I am much beholden to my supervisor Associate Professor Rudy Setiono. Prof Setiono has intrigued my interests in machine learning at first place. He guided me through the project with tremendous patience and timely responses. He taught me many scientific research methodologies, offered great help and advices when I encountered difficulties. He reviewed this report scrupulously and corrected many mistakes I made.

I thank Huynh Quang Thuan, Prof Setiono's former HYP student. He kindly provided me with literature resources and answered my queries from time to time.

I also thank Liu Ke Yao, an SOC alumnus. He consistently encouraged me to put my best to this project and taught me skills in report formatting.

List of Tables

Table 2.1: Description of Sequential Covering Approach.....	11
Table 2.2: Description of Learn_one_rule Approach	12
Table 2.3: Description of Learn_one_rule approach based on general to specific beam search with width k	14
Table 2.4: Description of AQ15 approach.....	16
Table 2.5: Description of Generate_STAR approach in AQ15	17
Table 2.6: Description of CN2 approach	18
Table 2.7: Description of Find_best_complex approach in CN2	20
Table 2.8: Description of X2R approach	22
Table 2.9: Description of ID3 approach	24
Table 3.1: Description of Chi2 discretization algorithm	27
Table 3.2: Description of GRG approach	29
Table 4.1: The data sets used in the experiment	36
Table 4.2: The result of the GRG experiment.....	37
Table 4.3: The comparison of the accuracy of GRG, C5.0, M5' and N2C2S.....	38
Table 4.4: Summary of the comparison result of GRG, C5.0, M5' and N2C2S.....	38
Table 4.5: The comparison of accuracies of GRG, CN2, AQ15 and human experts in three medical domains	40
Table 4.6: The comparison of complexities of GRG, CN2, AQ15 and human experts in three medical domains	40
Table 4.7: The comparison of the training and testing accuracy of GRG, CN2 and AQ15 in three medical domains	41
Table 4.8: The distribution of Breast Cancer samples according to their deg-malig and inv-nodes.....	42
Table 4.9: The updated distribution of Breast Cancer samples according to their deg-malig and inv-nodes after first iteration.....	43
Table 4.10: The updated distribution of Breast Cancer samples according to their deg-malig and inv-nodes after second iteration	44

Table 4.11: The updated distribution of Breast Cancer samples according to their deg-
malig and inv-nodes after third iteration..... 44

List of Figures

Figure 2.1: An example of general to specific beam search with width $k = 2$	13
Figure 3.1: Complete program flow of GRG approach	30

Contents

Chapter 1	Introduction	8
1.1	Background.....	8
1.2	Our Approach.....	8
1.3	Report Organization.....	9
Chapter 2	Related Works.....	10
2.1	Sequential Covering Algorithms.....	10
2.1.1	Learning Sets of Rules	10
2.1.2	The Sequential Covering Algorithm.....	10
2.1.3	General to Specific Beam Search	12
2.2	AQ Algorithms.....	15
2.2.1	The AQ Algorithm Family.....	15
2.2.2	AQ15.....	15
2.2.3	Summary of AQ.....	17
2.3	CN2 Induction Algorithm.....	18
2.3.1	The CN2 Algorithm	18
2.3.2	CN2's Find_best_complex.....	19
2.3.3	Summary of CN2	20
2.4	X2R Fast Rule Generator.....	21
2.5	ID3	22
2.5.1	Decision Tree Method.....	22
2.5.2	ID3	22
2.5.3	Entropy and Information Gain Measures.....	24
Chapter 3	Our Approach	26
3.1	Data Discretization.....	26
3.2	Greedy Rule Generation Algorithm.....	28
3.2.1	Description of the Algorithm.....	28

3.2.2	Program Flow.....	30
3.2.3	Notations Used in the Algorithm	31
3.2.4	A Detailed Description of the Algorithm.....	32
Chapter 4	Evaluation.....	35
4.1	Experimental Setup.....	35
4.2	Datasets Used in the Experiments.....	36
4.3	Results.....	36
4.4	Comparisons	37
4.4.1	Comparisons to Other Machine Learning Approaches.....	37
4.4.2	Comparisons to Other Rule List Generation Algorithms	39
4.4.3	A Case Study of GRG.....	41
Chapter 5	Conclusion	46
5.1	Summary	46
5.2	Future Work	46
Bibliography	48

Chapter 1

Introduction

1.1 Background

In our world, data are increasing dramatically in amount and usages, hence, to gather useful information from the ocean of data is becoming more important nowadays. There are many approaches to achieve this goal, and most of them are in the field of machine learning. Generating a set of rules from learning the data is an important method to do such data mining. Unlike the “black box” approaches—artificial neural networks, which takes the inputs, classifies them based on internal weights, and outputs the classification results, rule generation approach can output a decision tree, a decision rule list or a rule set that can be interpreted not only by machines but also by our human beings, and thus it uncovers the insights of each data set so that we can learn our world better. Approaches such as AQ (Michalski 1969, Michalski, Mozetic, Hong and Lavrac 1986), CN2 (Clark and Niblett, 1989), ID3 (Quinlan, 1983) and X2R (Liu and Tan, 1995) belong to the family of rule generation algorithm. They have achieved good results in some data sets, and some of the good results are comparable to or even outperform human experts. However, there are also limitations of these approaches. Complexities, including the time complexity and the size of result structure are often very high. In addition, as the simplicity and performance of the final learned rule set are usually dilemmas in these approaches, the improvement of the performance often cause the increase of the size of rule set and thus deteriorate the simplicity. These problems are unpleasant issues for these approaches.

1.2 Our Approach

In our research, we implemented the candidate approach Greedy Rule Generation algorithm (GRG) (Odajima, Hayashi, and Setiono, 200?) using Java and some classes

form WEKA (Holmes, Donkin and Witten, 1994). GRG is a sequential covering algorithm which generates one general rule per iteration. It differs from other rule generation algorithms mainly in the division of the examples space into subspaces and the rule merging process. To overcome the common problem of high complexity in rule generation algorithms, we added an upper limit to the subspace size and thus guaranteed small time complexity and rule list complexity. Chi2 discretization (Liu and Setiono, 1995) is used for attribute discretization, and information gain measures are used for feature selection. Because of the elimination of number of attributes and intervals of each attribute, the size of subspaces constructed based on these attributes is controlled under the upper limit. Through our intensive ten 10-fold testing of 14 data sets in the experiment, GRG has shown competitive performance in comparison with algorithms as M5' (Frank, Wang, Inglis, Holmes, and Witten, 1998) and N2C2S (Setiono, 2001) and better performance over rule generation algorithms as AQ (Michalski et al. 1986) and CN2 (Clark et al., 1989).

1.3 Report Organization

Chapter 1 gives the background of some major algorithms on rule generating and briefly introduces our approach to this problem. Chapter 2 describes some related work to our approach, as well as the algorithms and methods involved in designing and implementing our program. Chapter 3 explains the core algorithm of our approach. Chapter 4 gives the detail of our experiment set up, results and comparisons to other methods. Chapter 5 is the summary and conclusion of the ideas presented in this report.

Chapter 2

Related Works

2.1 Sequential Covering Algorithms

2.1.1 Learning Sets of Rules

In the domain of algorithms which learn sets of rules, there are two families: sequential covering algorithms and simultaneous covering algorithms. Algorithms like ID3 (Quinlan, 1983), for example, the rule generating process progresses in every tree branch at the same time, and hence it covers the examples simultaneously while generating the rules. Unlike simultaneous covering algorithms, a sequential covering algorithm only generates one rule per iteration when it processes through the entire example set. It conducts many iterations until the threshold is met, and eventually generates a set of rules. Sequential covering algorithms are among the most popular approaches to learn disjunctive rule set. The typical family members of sequential covering algorithms are the AQ algorithm (Michalski, 1969, Michalski et al. 1986), CN2 algorithm (Clark et al., 1989), and X2R algorithm (Liu et al., 1995). They share the common characteristic of learning a rule set by adding one rule at a time, though they differ in how the one rule is learnt. Sections 2.2, 2.3, 2.4 and 2.5 will describe some of these sequential covering algorithms as well as simultaneous covering algorithms, which relate to our greedy rule generation algorithm, in details.

2.1.2 The Sequential Covering Algorithm

The sequential covering algorithms perform greedy search formulating a sequence of rules without backtracking. It begins with an empty learned rule set, and then it goes through the examples to generate one best rule using `Learn_one_rule` procedure. It continuously and iteratively generates one rule and adds the rule to the set of learned

rules, until the threshold is met. It then sorts the list of learned rules according to certain criteria, and returns the list of sorted list of learned rules as the learning result. The prototypical sequential covering algorithm is described in Table 2.1.

```

Sequential_covering (Target_attribute, Attributes, Examples, Threshold)
    learned_rules ← {}
    rule ← Learn_one_rule (Target_attribute, Attributes, Examples)
    while (Performance (rule, Examples) > Threshold)
        learned_rules ← learned_rules + rule
        Examples ← Examples – {examples correctly classified by rule}
        rule ← Learn_one_rule (Target_attribute, Attributes, Examples)
    learned_rules ← sort learned_rules accord to Performance over Examples
    return learned_rules

```

Table 2.1: Description of Sequential Covering Approach

The Learn_one_rule method in sequential covering algorithms differs from approach to approach. Each approach of sequential covering family implements its own version of Learn_one_rule method. Though these different versions have different focuses and features, they share the same general algorithm structure to find the best rule. Learn_one_rule begins with the most general hypothesis as the initial *best_hypothesis*, and initializes *candidate_hypothesis* set to contain the *best_hypothesis*. It then starts the following iterative process to find the best rule. It specializes the *candidate_hypothesis* by adding one attribute-value pair to it. For example, the hypothesis (color = green) can be specialized by adding the attribute-value pair (height < 5) to a new hypothesis (color = green \wedge height < 5). It then makes these newly specialized hypotheses the *new_candidate_hypotheses*. It eliminates the useless hypotheses, usually those duplicate or inconsistent ones, in the *new_candidate_hypotheses*. It compares the hypotheses pairwise in *new_candidate_hypotheses* to find the *best_hypothesis* and makes it the new *candidate_hypothesis*. The above process is repeated until no more *best_hypothesis* can be found and the *candidate_hypothesis* becomes null. For examples, the process terminates when the *best_hypothesis* can no more be specialized. Finally, it finds the most frequent value of the *Target_attribute* (the class) and makes it the prediction. The

best_rule is built in the form “if *best_hypothesis* then *prediction*”. The description of the frame work of this Learn_one_rule method is shown in Table 2.2.

```

Learn_one_rule (Target_attribute, Attributes, Examples)
  best_hypothesis ← the most general hypothesis
  candidate_hypothesis ← {best_hypothesis}
  while (candidate_hypothesis is not empty)
    new_candidate_hypotheses ← specialize (candidate_hypothesis)
    new_candidate_hypotheses ← eliminate_useless (new_candidate_hypotheses)
    for (each h in new_candidate_hypotheses)
      if (h is better than best_hypothesis)
        best_hypothesis ← h
    candidate_hypothesis ← best_hypothesis
  prediction ← the most frequent value of Target_attribute of Examples covered by best_hypothesis
  best_rule ← if best_hypothesis then prediction
  return best_rule

```

Table 2.2: Description of Learn_one_rule Approach

2.1.3 General to Specific Beam Search

An effective way to implement learn-one-rule algorithm which is an important component of sequential covering algorithms is to use the general to specific beam search. Similar to the search fashion of ID3 algorithm, it expands the branch in the tree every step to find the best rule; however, there is a difference between the general to specific beam search and ID3 algorithm. Instead of developing a full subtree as ID3 does, general to specific beam search finds the descendant with the best performance and expands it. Because it only expands one descendant at a time, the width of the “beam” in the search of best candidates is one. When leaf nodes are reached, it finds the best leaf node according to its performance and makes it the rule. There are many ways to define the measure of performance of a descendant. Among them, the most popular one is the entropy measure, which means the descendant with the lowest entropy is seen as having the best performance. We introduce this entropy measure in more detail in Section 2.5.3.

The general to specific search is in nature a greedy depth-first search with no backtracking, which is fast with relatively good performance especially when applied to a large amount of data; however, it may also unfortunately result in a suboptimal choice. To compensate for this disadvantage, the practice of using beam search with width k greater than one can be adopted. Instead of following the best single descendant in the tree each time, it can now follow the best k descendants at the same time, and therefore reduce the danger of falling in a suboptimal option.

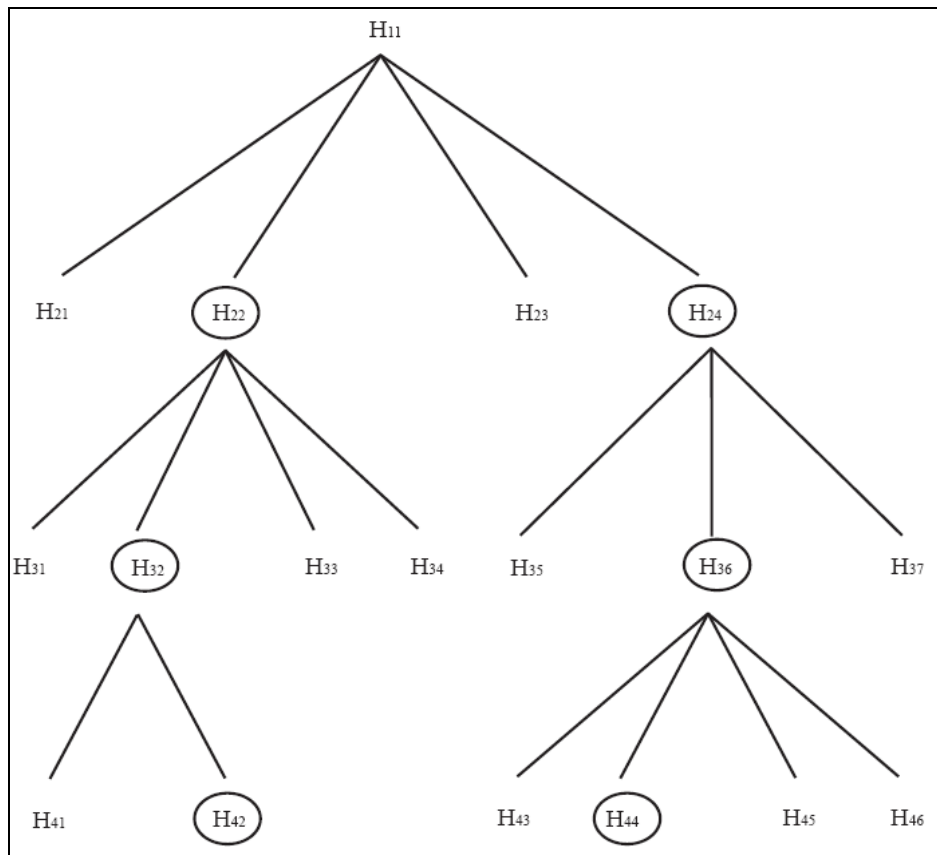


Figure 2.1: An example of general to specific beam search with width $k = 2$

An example of the general to specific beam search with width $k = 2$ is illustrated in Figure 2.1. The search starts at the most general hypothesis H_{11} , specializes it to form four new hypotheses H_{21} , H_{22} , H_{23} , and H_{24} , which are also the descendants of H_{11} . Then it chooses two best hypotheses (because the width of the beam is two) H_{22} and H_{24} and again specializes them to form seven new hypotheses H_{31} , H_{32} , H_{33} , H_{34} , H_{35} , H_{36} , and

H₃₇. The best two of the seven hypotheses H₃₂ and H₃₆ are chosen and again specialized. When the six new hypotheses are formed, the best two of them H₄₂ and H₄₄ are chosen for further specialization. However, since H₄₂ and H₄₄ are already the leaf nodes and they have no more descendants to specialize, the iterative process of the beam search terminates here.

The Learn_one_rule procedure using the general to specific beam search with width k greater than one is almost the same as the Learn_one_rule using beam search with width of one, except that it takes the width k as an argument, and keeps finding the best k hypotheses in the *new_candidate_hypotheses* and make them the *candidate_hypotheses* for next iteration. This version of the Learn_one_rule procedure is described in Table 2.3.

```

Learn_one_rule (Target_attribute, Attributes, Examples,  $k$ )
  best_hypothesis ← the most general hypotheses
  candidate_hypotheses ← {best_hypothesis}
  while (candidate_hypotheses is not empty)
    new_candidate_hypotheses ← specialize (candidate_hypotheses)
    eliminate_useless (new_candidate_hypotheses)
    for (each  $h$  in new_candidate_hypotheses)
      if ( $h$  is better than best_hypothesis)
        best_hypothesis ←  $h$ 
    candidate_hypotheses ← the  $k$  best members of new_candidate_hypotheses
  prediction ← the most frequent value of Target_attribute of Examples covered by best_hypothesis
  best_rule ← if best_hypothesis then prediction
  return best_rule

```

Table 2.3: Description of Learn_one_rule approach based on general to specific beam search with width k

2.2 AQ Algorithms

2.2.1 The AQ Algorithm Family

A family of algorithms called AQ (Michalski 1969, Michalski et al. 1986) is a variation of sequential covering algorithm. AQ uses optional background knowledge that consists of rules already known to learn rules incrementally from positive examples and negative examples. For each decision class, the examples belonging to the class are positive examples of the class, and all other examples are negative examples.

Rules in AQ are described by selectors, complexes and covers. A selector is a relational statement of the form (term relation reference), where term is a variable or a conjunction of terms, a relation is one of the relation symbols $\{<, <=, =, >=, >, <>\}$, and a reference is a value or a disjunction of values. For example, (color = green) is a selector, (color = green \vee red) is also a selector. A complex in AQ is a conjunction of selectors. For instance, (color = green \vee red) \wedge (height < 5) is a complex. On the other hand, a cover in AQ is a disjunction of complexes. The cover ((color = green \vee red) \wedge (height < 5)) \vee ((color = green \vee yellow) \wedge (height = 10)) consists of two complexes.

2.2.2 AQ15

The AQ15 inductive learning program is a descendant of the GEM program (“Generalization of Examples by Machine”) and the AQ1-AQ11 series of inductive learning programs. Like its ancestors, AQ15 is also based on the AQ algorithm (Michalski 1969), which incrementally generates decision rules from examples. Similar to other sequential covering algorithms, when building a rule, AQ15 performs a heuristic search through the example space. However, unlike other approaches which find best rules to cover all the classes, AQ15 finds best rules (represented by *cover* in the original paper) for each class in separate processes.

For each class c present in the input space, AQ15 divides the examples into two sets: E^+ , consisting of the positive examples which are classified to class c ; and E^- , consisting of the negative examples which are classified to other classes rather than class c . The *cover*, the best rule list, is initialized to be empty set. AQ15 then starts the incremental rule

generation process iteratively. It first selects an example e from the positive example set $E+$ either randomly or according to optional background knowledge known in advance. It then generates a set of candidate complexes $STAR$ (as the name first introduced in the original paper) according to this selected positive example e , the positive example set $E+$ and the negative example set $E-$. Next it chooses the *best_complex* from $STAR$ according to user defined criteria and adds the *best_complex* to *cover*. Those positive examples that are covered by this *best_complex* are then removed from $E+$. AQ15 repeats the process until the positive example set $E+$ becomes empty, i.e. the positive examples in $E+$ have all been covered by the *best_complexes* in the *cover* and have therefore been removed from $E+$. When the *covers* for every class in *Examples* are generated by the above process, it returns them as the learned rule list. A description of AQ15 approach is in Table 2.4.

```

AQ15 (Examples, Background_knowledge, Criteria)
  for (each class  $c$  in Examples)
     $E+ \leftarrow$  {positive examples of  $c$  in Examples}
     $E- \leftarrow$  {negative examples of  $c$  in Examples}
    cover  $\leftarrow$  {}
    while ( $E+$  is not empty)
       $e+ \leftarrow$  Select_example( $E+$ , Background_knowledge)
       $STAR \leftarrow$  Generate_STAR ( $E+$ ,  $E-$ ,  $e+$ )
      best_complex  $\leftarrow$  Find_best_complex ( $STAR$ , Criteria)
      cover  $\leftarrow$  cover + best_complex
       $E+ \leftarrow E+ -$  {examples covered by best_complex}
  return cover for each class  $c$ 

```

Table 2.4: Description of AQ15 approach

The generating of candidate complex set `Generate_STAR` is the core procedure that leads to the *best_complex* and *cover*. $STAR$, the candidate complex set, is initialized to be empty. While the negative example set $E-$ is not empty, it repeats the following process to trim the size of $STAR$. It selects an example $e-$ from $E-$, if $e-$ happens to be included in $STAR$, i.e. $e-$ is one the complex in the candidate complex set $STAR$, then it creates another complex set *NewSTAR* which consists of those complexes that cover $e+$ (the

positive example passed as an argument to Generate_STAR) but not e^- . It then creates an intersection of $STAR$ and $NewSTAR$, and makes the result of this intersection the updated $STAR$. This intersection is the trimming of the candidate complex set, which deletes those inconsistent complexes covering any negative examples and only retains those consistent ones covering only positive examples. Table 2.5 shows the description of this procedure.

```

Generate_STAR( $E^+$ ,  $E^-$ ,  $e^+$ )
   $STAR \leftarrow \{\text{example space}\}$ 
  while ( $E^-$  is not empty)
     $e^- \leftarrow$  an example from  $E^-$ 
    if ( $e^- \in STAR$ )
       $NewSTAR \leftarrow \{\text{complexes that cover } e^+ \text{ but not } e^-\}$ 
       $STAR \leftarrow STAR \cap NewSTAR$ 
  return  $STAR$ 

```

Table 2.5: Description of Generate_STAR approach in AQ15

2.2.3 Summary of AQ

As an early approach in machine learning, AQ created a new way in rule list generation. It is similar to the sequential covering approaches; however, it still differs from them in some ways. First, instead of finding rules covering all classes, it explicitly seeks rules that cover only a particular class at one time. The final rule set for all classes is the disjunction of the covering rules for each class. Second, though AQ also uses general to specific beam search for each rule as other approaches using the Learn_one_rule prototypical procedure, it uses a single positive example as a seed to start the search. It considers only those attributes (selectors) satisfied by the positive example as it progresses and specializes more hypotheses (complexes).

Because of the strict matching of complexes and positive examples as well as the strict exclusion of negative examples, AQ's final rule set is often large. This stands as an even bigger disadvantage because the rule set is not strictly speaking a rule list in the sense that the rules in the set are not ordered. Thus, without the labeling of priority of each rule, it may become difficult to search for a certain rule to predict a new example. Noise data can

also easily confuse AQ due to this tight matching. Any misclassified examples or inaccurate measurements of attributes can result in a set of misleading rules.

2.3 CN2 Induction Algorithm

2.3.1 The CN2 Algorithm

As a member of sequential covering algorithm family, CN2 finds only one rule per iteration and inserts it to its final rule list. Rules induced by CN2 have the form “if *complex* then *class*”, where the *complex* is a conjunction of attribute-value pairs. For example, “color = green” is a complex where “color” is the attribute and “green” is the value. Hence, “if color = green, then grass” is a rule, where “color = green” is the complex and “grass” is the class.

CN2 initializes its learned rule list to be empty, then finds the best rule by using the Find_best_complex procedure to find the best complex, forms a best rule using the best complex and the most common class occurred in the examples covered by the best complex, removes the examples covered by the best complex and adds the best rule into the learned rule list. It repeats the above process until no best complex can be found by the Find_best_complex procedure or the examples have all been removed because they have all been covered by those best complexes found in each iteration. The CN2 approach is also shown in Table 2.6.

```

CN2 (Examples, Attributes, Threshold)
  learned_rules ← {}
  while (best_complex is not nil and Examples is not empty)
    best_complex ← Find_best_complex (Examples, Attributes, Threshold)
    Examples' ← {examples covered by best_complex}
    best_rule ← if best_complex then most_common_class of Examples'
    Examples ← Examples – Examples'
    learned_rules ← learned_rules + best_rule
  return learned_rules

```

Table 2.6: Description of CN2 approach

2.3.2 CN2's Find_best_complex

The procedure Find_best_complex employed in CN2 approach uses the pruned general to specific beam search algorithm with width k to keep the set of best specialized rules. It starts with an empty candidate hypotheses set *STAR* (the name *STAR* is inherited from the AQ algorithms as CN2 is developed based on AQ) and a nil *best_complex*. It then carries out the iterative process to look for the *best_complex*.

Each complex in *STAR* is specialized by adding the attributes that have not been included in the complex yet. The newly specialized complexes are grouped in a new candidate hypotheses set *NewSTAR*. The null hypotheses and the hypotheses already included in *STAR* are removed from *NewSTAR*, then a pair-wise comparison is conducted among the complexes in *NewSTAR*, and the best one is set as *best_complex*. The heuristics used to determine the best complex in CN2 approach are the information-theoretic entropy measure and the likelihood ratio statistic significance test (Kalbfleish, 1979). The former evaluates whether a complex is good, while the latter evaluates whether a complex is reliable. After choosing the current *best_complex*, the size of *NewSTAR* is checked. If it is greater than the user defined *Threshold*, then the complex with worst performance in *NewSTAR* is found and removed from *NewSTAR*. The elimination of the worst performed complex is repeated until the size of *NewSTAR* is no greater than the *Threshold*. *NewSTAR* then becomes the *STAR* for next iteration.

The iterative process terminates when the candidate hypotheses set *STAR* becomes empty, i.e. no more complex can be further specialized. The procedure then returns the *best_complex* as the result. The detailed Find_best_complex procedure is shown in Table 2.7.

```

Find_best_complex (Examples, Attributes, Threshold)
  STAR ← {empty_complex}
  best_complex ← nil
  while (STAR is not empty)
    for (each Attribute y)
      NewSTAR ← NewSTAR + {STAR, Attributes}
      NewSTAR ← NewSTAR – STAR – {null complexes}.
    for (every complex Ci in NewSTAR)
      if (Ci is better-than best_complex)
        best_complex ← Ci
    while (|NewSTAR| > Threshold)
      NewSTAR ← NewSTAR – worst-complex-in-NewSTAR
  STAR ← NewSTAR
  return best_complex

```

Table 2.7: Description of Find_best_complex approach in CN2

2.3.3 Summary of CN2

The development of CN2 is based on the previous AQ approach. However, it differs from AQ in several ways.

First, the result of CN2 is an ordered learning rule list, unlike the unordered one in AQ. Rules in the list are labeled with priorities. When applying these rules to a new example, the rule with highest priority is checked first to see if the example can be classified using this rule. If not, other rules are checked according to their priorities. Thus the classifying process of a new example is faster than AQ approach.

Second, because CN2 does not build best rules to cover every single positive example and excludes every single negative example as AQ does, noise will not confuse CN2 as much as AQ. Also, the problem of overfitting of training data is not severe in CN2.

2.4 X2R Fast Rule Generator

X2R (Liu et al, 1995) is a relatively newer approach compared to AQ and CN2. It utilizes discretization, feature selection, and concept learning to generate explicit production rules from raw data. Since our algorithm also uses discretization and feature selection, the details of these data preprocessing techniques will be given in Chapter 3.

The X2R algorithm first initializes the learned rule list *learned_rules* to be empty, and sorts the examples according to example frequency, where an example here is similar to a complex in AQ and CN2 approach. Then X2R picks the most frequent occurring example as the base to generate a *best_rule* and adds the *best_rule* to the rule list *learned_rules*. It then finds all the examples that are covered by the *best_rule* and removes them from the example space *Examples*. It repeats the above process iteratively and continuously adds the generated *best_rules* to the learned rule list *learned_rules* until the examples space *Examples* becomes empty because all data examples have been covered by the rules generated and they have all been removed.

Next, X2R clusters the learned rules according to their class labels, i.e. rules of the same class are clustered together as one group of rules. In each of these clusters, more than one rule may cover the same example. For examples, the rule “if (color = green) and (height < 5) then grass” is already contained in a bigger rule “if (color = green) then grass”, and thus the rule “if (color = green) and (height < 5) then grass” is redundant. X2R eliminates these redundant rules in each cluster to further reduce the size of the best rule list *learned_rules*. In each cluster, there are probably also some examples not covered by any of the rules already generated. In this case, X2R chooses a default rule for these example so that when a new example is not covered by any of the generated rules, the default rule will be applied to it and classify it. The description of X2R approach is in Table 2.8.

```

X2R(Examples)
  learned_rules ← {}
  Sort_on_frequency (Examples)
  while (Examples is not empty)
    best_rule ← pattern occurred most frequently
    learned_rules ← learned_rule + best_rule
    Examples ← Examples – {patterns covered by best_rule }
  Cluster_according_to_class(learned_rules)
  Eliminate_redundant(learned_rules)
  Determine_default_rule(learned_rules)
  return learned_rules

```

Table 2.8: Description of X2R approach

2.5 ID3

2.5.1 Decision Tree Method

Decision tree learning is one of the most popular and practical approaches in machine learning. Unlike sequential covering algorithms discussed in previous sections, decision tree learning does not learn one rule at a time, instead it learns the entire set of disjunction of rules simultaneously, and therefore decision tree learning algorithms are categorized as simultaneous covering algorithms. The result of decision tree learning is also a set of rules like other sequential covering algorithms, but it is in the form of a tree of rules as its name suggests, not an ordered list of rules. The family of decision tree algorithms includes ID3 (Quinlan, 1986), ASSISTANT (Cestnik, 1986), C4.5 (Quinlan, 1993) and C5.0 (Quinlan, 1998). As C4.5 and C5.0 are successors of ID3, we will use ID3 to represent this family of algorithms and introduce it in the following section.

2.5.2 ID3

The ID3 decision tree building is a recursive process that starts with an empty tree root. Before applying the recursion, it first checks whether the termination condition is met. The termination conditions include the homogeneity of *Examples* and the empty of *Attributes*. If all examples in *Examples* have the same *Target_attribute* (the same class), the root is labeled with the value of this *Target_attribute* (class) and returned. Also, if the

set *Attributes* is empty, which means all the attributes in *Attributes* have already been used as classifiers for the decision tree building and no more free attribute is available for further classification, then root is labeled with the most common value of *Target_attribute* in *Examples* and returned. Otherwise, the *Examples* is not homogenous and the set *Attributes* is not empty, thus the tree can be further extended for more classification.

When extending the decision tree, an attribute *A* that best classifies *Examples* is selected among the *Attributes*. There are different methods of attribute selection, ID 3 uses entropy measures and information gain measures (see section 2.5.3). For each possible value v_i of the selected attribute *A*, a corresponding *new_branch* $A = v_i$ is then built under *root*. The examples in *Examples* are divided into subsets of examples according their value of attribute *A*. $Examples_{v_i}$ is the subset of examples for the branch $A = v_i$. If $Examples_{v_i}$ is empty, i.e. there is no such example that has the value v_i for *A*, then the *new_branch* $A = v_i$ is labeled with the most common value of *Target_attribute* in *Examples*. Otherwise, the *new_branch* will be built using recursion with *Target_attribute*, $Attributes - \{A\}$ and $Examples_{v_i}$ as its arguments.

When all branches have been built by calling ID3 function recursively with its own version of arguments, the decision tree building process terminates and the *root* is returned as the final result. The description of the ID3 approach is in Table 2.9.

```

ID3 (Target_attribute, Attributes, Examples)
  root ← nil
  if (Examples has only one class c)
    Label (root, c)
    return root
  else if (Attributes is empty)
    Label (root, most common value of Target_attribute in Examples)
    return root
  else
    A ← Find_best_attribute (Attributes, Examples)
    for (each value  $v_i$  of A)
      new_branch ← (A =  $v_i$ )
      root ← Add_new_branch (new_branch)
      Examples $v_i$  ← {examples in Examples that have value  $v_i$  for A}
      if (Examples $v_i$  is empty)
        Label (new_branch, most common value of Target_attribute in Examples)
      else
        ID3 (Target_attribute, Attributes - {A}, Examples $v_i$ )
    return root

```

Table 2.9: Description of ID3 approach

2.5.3 Entropy and Information Gain Measures

Some data sets contain large number of attributes, but not all of them are equally useful at classifying the training examples; therefore, we need a quantitative measure to compare the usefulness of the attributes. The statistical property information gain, which gives a measure on how much an attribute improves the classification of the examples according to the target attribute (class), is used for the comparison of attributes. Information gain calculation in ID3 is based on entropy, the measure of impurity of a data set. If the examples are less heterogeneous, its entropy value is small. In the case of homogeneity (all examples have the same target attribute value or same class), the entropy value is zero. On the other hand, if the data set consists of many different target attribute values or classes of nearly equal proportion, the entropy value of the data set is very large. The entropy of a data set S is calculated as follows:

$$Entropy(S) \equiv \sum_{i=1}^k -p_i \log_2 p_i ,$$

where p_i is the proportion of data set S classified to class i , and k is the number of classes.

For an attribute A , its information gain is calculated as:

$$Gain(S, A) \equiv Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

where $Values(A)$ are the possible values of attribute A , S_v is the subset of S in which the attribute A has the value v . The entropy of each S_v is first calculated, and the sum of weighted entropies is taken, where the weight for each entropy of S_v is the ratio of number of examples in S_v to the number of the total number of examples in data set S . Then the information gain of attribute A equals the entropy of S minus the weighted sum of entropies for all S_v . Thus the information gain refers to the deduction of the entropy as well as the impurity of the data set by selecting attribute A as its classifier. The larger the information gain A has, the more useful A is.

Chapter 3

Our Approach

3.1 Data Discretization

Data values are normally categorized as numerical values (also known as continuous values, as “1.5”, “300” etc), and ordinal values (as “small”, “medium” and “large” etc). The latter two types of data are also discrete data, whereas the former is not. Since our aim is to generate rules from discretized data, we have to discretize the numerical data before start applying the algorithm.

The discretization method we adopt here is Chi2 discretization (Liu and Setiono, 1995) (Chi2 reads as “kai square”, sometimes also noted as χ^2). First we initialize the significance levels of all attributes to be 0.5. Then while there are mergeable attributes in *Attributes*, the iterative process of merging starts. For each mergeable attribute i in the attribute set *Attributes*, the data entries in *Examples* are sorted into intervals according to their values of attribute i . For every two adjacent intervals, Chi2 values are initialized using the Chi2 statistic formula (see Equation 3.1). Then iterative process of merging intervals for attribute i starts. The pair of adjacent intervals that have the minimum Chi2 values among all adjacent interval pairs is merged to one interval. The Chi2 values are then recalculated after the merging, and another pair of adjacent intervals with minimum Chi2 values is chosen to be merged for next round. The merging of the intervals terminates when no more intervals are further mergeable. The inconsistency rate of the data set with respect to attribute i is calculated and checked. If the inconsistency rate is less the upper limit δ defined by user, the significance level of attribute i is decreased and the iterative merging process starts again from the initialization of the Chi2 values for

each adjacent interval pairs according to the new significance level. When all the merging process is done for each attribute, the Chi2 discretization finishes.

A description of Chi2 discretization algorithms can be found in Table 3.1. Functions $\text{Mergeable}(\text{Attributes})$ and $\text{Mergeable}(i)$ examine the mergeability of the *Attributes* and the attribute *i* respectively. $\text{Mergeable}(\text{Attributes})$ returns true if some attribute *i* in *Attributes* satisfies $\text{Mergeable}(i)$. And the “mergeability” of an attribute *i* is true as long as there exist two intervals whose values of attribute *i* can be merged into be one interval while the inconsistency rate of the merged data set *Examples* does not exceed a user defined limit δ . After merging, in some intervals, the data entries belonging to this interval may not have the same class labels. Some data entries with one class label are smaller in number compared to other data entries with other class labels, then the former are counted as inconsistent instances. The inconsistency rate for an attribute *i* is the ratio of the number of all inconsistent instances in all intervals regarding to attribute *i* to the total number of data entries in *Examples*.

```

Chi2_discretization(Attributes, Examples,  $\delta$ )
  for (each attribute i)
     $\text{sigLevel}(i) \leftarrow 0.5$ 
  while ( $\text{Mergeable}(\text{Attributes})$ )
    for (each i)
      if ( $\text{Mergeable}(i)$ )
        Sort(i, Examples);
        Chi2_initialization (i, Examples);
        do
          Chi2_calculation (i, Examples);
          while ( $\text{Mergeable}(i)$  is true)
            if ( $\text{Inconsistency}(i, \text{Examples}) < \delta$ )
               $\text{sigLevel}(i) \leftarrow \text{decreSigLevel}(\text{sigLevel}(i));$ 
            else
              Mergeable (i)  $\leftarrow$  false

```

Table 3.1: Description of Chi2 discretization algorithm

The Chi2 statistic formula used in the algorithm is

$$\chi^2 = \sum_{i=1}^2 \sum_{j=1}^k \frac{(A_{ij} - E_{ij})^2}{E_{ij}}, \quad (\text{Equation 3.1})$$

where k denotes the number of classes, A_{ij} denotes number of patterns in the i th interval and j th class, and E_{ij} denotes the expected frequency of A_{ij} . E_{ij} is calculated as:

$$R_i \times C_j / N,$$

where

$$R_i = \sum_{j=1}^k A_{ij}, C_j = \sum_{i=1}^2 A_{ij}, N = \sum_{i=1}^2 R_i.$$

We use these formulae to calculate the Chi2 values for the intervals in the Chi2 discretization algorithm, as described in Table 3.1.

3.2 Greedy Rule Generation Algorithm

3.2.1 Description of the Algorithm

GRG first initialize the `optimal_rule_list` (used to contain the final learnt rules) to be empty. It then builds subspaces according the attributes selected when preprocessing the data set. The subspaces have the dimension of $N_1 \times N_2 \times \dots \times N_j$, where j is the number of attributes selected and N_j is the number of possible values of attribute j . It sorts the examples into corresponding subspaces according to their attribute values. It then counts the frequencies of each class occurrences in each subspace and labels the subspaces with the class that has the maximum frequency. Each of the subspaces now has its own local rule R_0 of the form: if (attribute-value pairs of this subspace) then (label of this subspace).

After the subspaces are labeled, it starts the iterative rule generation process. For each iteration, it first initializes a new `rule_list` to be empty. It then selects the rule that covers the most examples among all local rules R_0 held by each subspace as a seed rule, and sets it as \bar{R} . From all the local rules, those with the same class as \bar{R} or “unlabeled” are selected and added to the `rule_list` together with \bar{R} . All the mergeable pairs of rules in the

rule_list are then merged into “bigger” rules and added to the rule_list, and the merged rules are again merged into even “bigger” rules and added until no more rules can be further merged. A best rule R^* is then selected from the rule_list according to four criteria (see Subsection 3.2.4). The subspaces are relabeled: the subspaces with the same class label as R^* is set “unlabeled” and their class frequencies are set to zero. R^* is added to the optimal_rule_list. The above rule generation process terminates when all the subspaces are finally labeled as “unlabeled”, i.e. they have been all covered by the rules generated. Table 3.2 shows each step in this brief description.

<pre> GRG(<i>Examples, Attributes</i>) <i>optimal_rule_list</i> ← {} <i>subspaces</i> ← Build_subspaces(<i>Examples, Attributes</i>) <i>subspaces</i> ← Label_subspaces(<i>Examples, Attributes</i>) while (∃ <i>subspace s</i> such that Label(<i>s</i>) ≠ “unlabeled”) <i>rule_list</i> ← {} \bar{R} ← the rule covers most examples in <i>subspaces</i> <i>rule_list</i> ← \bar{R} + {other rules R_0 such that the class of R_0 is the same as \bar{R} or “unlabeled”} while (Mergeable(<i>rule_list</i>)) <i>rule_list</i> ← <i>rule_list</i> + Merge_rule(<i>rule_list</i>) R^* ← Find_best_rule(<i>rule_list</i>) <i>subspaces</i> ← Relabel_subspaces(<i>Examples, Attributes, R^*</i>) <i>optimal_rule_list</i> ← <i>optimal_rule_list</i> + R^* return <i>optimal_rule_list</i> </pre>

Table 3.2: Description of GRG approach

3.2.2 Program Flow

A complete program flow is shown in Figure 3.1.

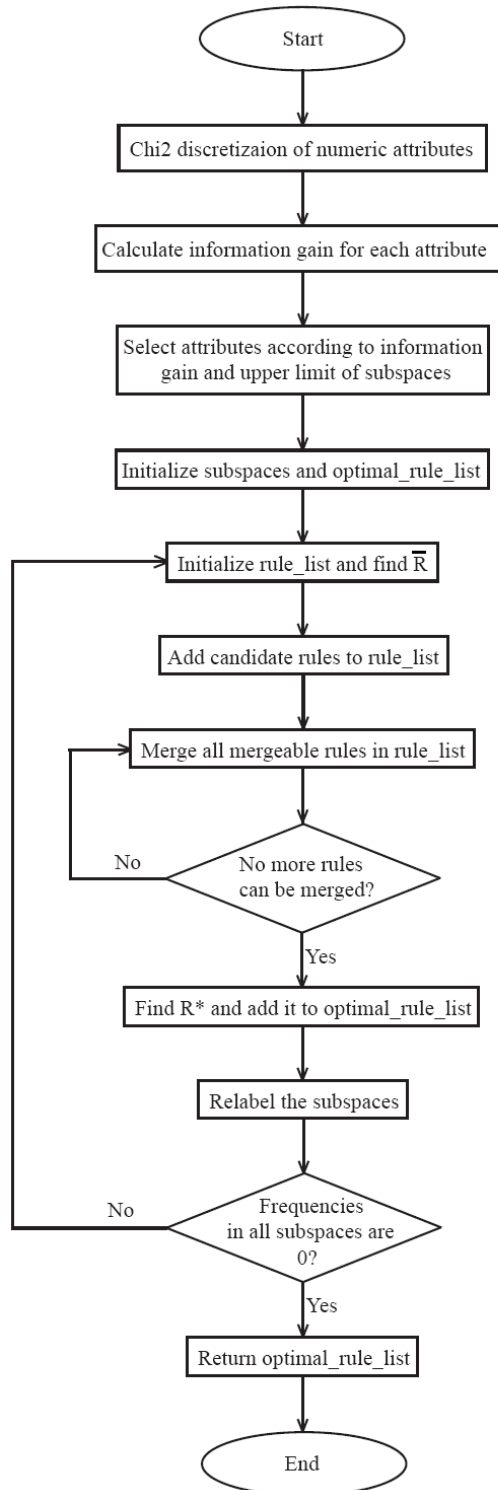


Figure 3.1: Complete program flow of GRG approach

3.2.3 Notations Used in the Algorithm

The notations used in our approach are defined as follows:

1. A discrete attribute A_j can have N_j possible values: $A_{j(1)}, A_{j(2)}, \dots, A_{j(N_j)}$,
2. A rule R with J conditions has the following form: $R = (RC_1, RC_2, \dots, RC_J, \Rightarrow y_i)$,
where
 - RC is the rule condition involving one or more values of attribute A_j . It has the following disjunctive form: if A_j is equal to $A_{j(j_1)}$ or A_j is equal to $A_{j(j_2)}$ or A_j is equal to $A_{j(j_3)}$ or \dots , or A_j is equal to $A_{j(j_m)}$ where $j_l \in \{1, 2, \dots, N_j\}$, $l = 1, 2, \dots, m$.
 - y_i is the rule conclusion for the samples that satisfy the condition of rule R , i.e. the samples are classified as members of class i .
3. n_j is the number of disjunctions in rule condition RC_j , $0 \leq n_j \leq N_j$. An attribute A_j is considered irrelevant for this rule if $n_j = N_j$.
4. Mergeable rules.

Consider two rules $(R_1C_1, R_1C_2, \dots, R_1C_J) \Rightarrow y_1$ and $(R_2C_1, R_2C_2, \dots, R_2C_J) \Rightarrow y_2$.

If the following two conditions:

- Condition 1. $y_1 = y_2$ or $y_1 = \text{"unlabeled"}$ or $y_2 = \text{"unlabeled"}$, and
- Condition 2. There exists exactly one index \hat{i} such that $R_1C_{\hat{i}} \neq R_2C_{\hat{i}}$

Are satisfied, then they are mergeable. The new rule is $R' = (R'C_1, R'C_2, \dots, R'C_{\hat{i}}, \dots, R'C_J) \Rightarrow y_i$, where

- $R'C_{\hat{i}} = R_1C_{\hat{i}} \cup R_2C_{\hat{i}}$, and
- The rule conclusion is y_i is

$$y_i = \begin{cases} y_1 & \text{if } y_i \neq \text{"unlabeled"} \\ y_2 & \text{if } y_i \neq \text{"unlabeled"} \\ \text{"unlabeled"} & \text{otherwise} \end{cases}$$

If the attribute A_i is an ordinal discrete variable where $A_{i(1)} < A_{i(2)} < \dots < A_{i(N_i)}$,

then in addition to the above 2 conditions, we also impose

- Condition 3. The values of this attribute in the two rule conditions $R_1C_{\hat{i}}$ and $R_2C_{\hat{i}}$ must be consecutive, that is, $R_1C_{\hat{i}}$ involves the value $A_{\hat{i}(k)}$ and the rule condition $R_2C_{\hat{i}}$ involves the value $A_{\hat{i}(k+1)}$, for some value of k . For the condition of the merged rule R' , these two attribute values are merged into a single value as well, namely $A_{\hat{i}(k)} \cup A_{\hat{i}(k+1)}$.

We are now ready to state our rule generating algorithm.

3.2.4 A Detailed Description of the Algorithm

GRG takes labeled data with discrete-valued attributes as its inputs and outputs an ordered list of classification rules. Let J be the number of attributes, and N_j be the number of discrete values of attributes A_j , $j = 1, 2, \dots, J$. In detail, the GRG algorithm works in the following steps.

1. Initialization.

- a. Decompose the input space into

$$S = \prod_{j=1}^J N_j$$

subspaces. An upper limit is set for the size of S due to the exponential time complexity, and only the attributes with relatively higher information gains are selected for the decomposition of the input space.

- b. For each of these subspaces S_p , $p = 1, 2, \dots, S$:

- Generate the rule R_p with J conditions ($R_pC_1, R_pC_2, \dots, R_pC_J$), where R_pC_J involves exactly one value of attribute A_j .
- Count the frequency of samples of class i in subspace S_p . Denote this frequency as $F_{p,i}$ and let

$$F_{p,\hat{i}} = \max_i F_{p,i}$$

- If $F_{p,\hat{i}} > 0$, then let the conclusion of rule R be $y_{\hat{i}}$. Otherwise, denote the class label for this subspace as “unlabeled”.

- c. Initialize the optimal rule set R to be empty.

2. Rule generation.
- a. Let \bar{p} and \bar{i} be such that

$$F_{\bar{p},\bar{i}} = \max_{p,i} F_{p,i}$$

and \bar{R} be the rule for the samples in subspace $S_{\bar{p}}$. If $F_{\bar{p},\bar{i}} = 0$, stop.

- b. Generate a list of rules consisting of
- i. \bar{R} , and
 - ii. all other rules R_0 such that $y_0 = y_{\bar{i}}$ or $y_0 = \text{“unlabeled”}$.
- c. For all mergeable pairs of rules in the generated list b, add the merged rule to the list and compute the class frequency of the samples covered by the new rule accordingly. Repeat this step until no new rule can be added to the list by merging.
- d. Among all the rules in the list, select the best rule R^* using the conditions (in decreasing importance):
- i. It must include \bar{R} ,
 - ii. It covers the maximum number of samples with the correct label,
 - iii. It has the highest number of irrelevant attributes,
 - iv. It covers the largest subspace of the input.
- e. Let $R := R \cup R^*$
- f. Set the class label of all samples in the subspaces covered by rule R^* to “unlabeled” and their corresponding frequencies to 0.
- g. Repeat from Step 2a.

In step 2.d, the conditions on how R^* is chosen is explained here. The best rule must include \bar{R} as the rule being generated must classify the samples with the maximum frequency in the subspace as defined in 2a. Due to merging of a pair of rules in Step 2b, it is possible that some rule added to the list do not cover samples classified by \bar{R} . The second condition is that the rule must cover the highest number of training data samples as we are generating ordered rules, rules that cover more samples are given higher importance. The third condition is to select a rule with the highest number of irrelevant attributes so that simpler rules involving fewer attributes will be generated. Finally, if

there is a tie in the rule that satisfies all the three conditions, the rules that cover the largest input subspace will be added to the optimized rule set.

Chapter 4

Evaluation

4.1 Experimental Setup

In order to compare the effectiveness of the Greedy Rule Generation method, 18 data sets were selected. For each data set, the experimental setting was identical as follows:

1. Ten-fold cross-validation scheme: each data set was divided into ten subsets of equal size randomly. Nine of them were used for training, and the other was used for measuring the predictive accuracy of rules generated from the algorithm. This process was conducted ten times for each data set so that each subset was used as a test set once. The average test set accuracy of the ten sets of rules was reported as the ten-fold accuracy. To minimize the variance of ten-fold accuracy (e.g. due to uneven distribution of data or random partitioning of data into subsets), ten 10-fold cross validation was used in the actual experiment.
2. Chi2 discretization: the subsets used for training were discretized by Chi2 method before applying the Greedy Rule Generation algorithm, so that the subspaces for data were determined. The inconsistency rate of Chi2 discretization was set to 0.05 for every data set.
3. Information gain: entropy and information gain for each attribute in a data set (whose attributes contain only continuous integer values) was calculated as the heuristic of attribute selection. The attributes were then rearranged in ascending order of their information gain values in order to minimize the total number of subspaces and thus lessen the computation complexity.
4. Missing Values: a missing continuous attribute value was replaced by the average of the non-missing values. A missing discrete attribute value was assigned the value “unknown”.

4.2 Datasets Used in the Experiments

For this experiment we used 14 datasets, which are data abstracted from real world classification problems. The detail information of these data sets is listed in Table 4.1. These machine learning data sets are available at UCI Machine Learning Repository (Blake and Merz, 1998). The following data sets were selected because their tested accuracy of other algorithms are also available, which is convenient for comparison purpose.

Dataset	Size	Missing Values (%)	Attributes			Num of Classes
			Continuous	Binary	Nominal	
australian	690	0.6	6	4	5	2
breast-cancer	286	0.0	0	3	6	2
breast-w	699	0.3	9	0	0	2
german	1000	0.3	6	3	4	2
glass (G2)	163	0.0	9	0	0	2
heart-c	302	0.2	6	3	4	2
heart-h	294	20.4	6	3	4	2
heart-statlog	270	0.0	13	0	0	2
horse-colic	368	23.8	7	2	13	2
hepatitis	155	5.6	6	13	0	2
iris	150	0.0	4	0	0	3
lymphography	148	0.0	0	9	6	4
primary-tumor	339	3.9	8	3	5	21
sick	3772	5.5	7	20	2	2

Table 4.1: The data sets used in the experiment

4.3 Results

The whole program, including the GRG algorithm, Chi2 discretization, information gain calculation is written in Java. It was run on personal desktop with Intel Pentium 4 3.0GHz CPU and Kingston 512MB RAM. Some data sets have been run with very little time because their relatively smaller size and fewer attributes, whereas some other data sets large in size and with more attributes experienced difficulty to finish the running before the computer exhausted its memory. That is the major reason why the upper limit for the number of subspaces came into use.

The results of the run of the program on selected data sets are summarized in Table 4.2. The ten 10-fold accuracy is the average of ten runs of 10-fold cross validation. The number of rules is the size of the optimal rule list generated by GRG algorithm. Number of useful attributes is the number of attributes selected to decompose the input space to subspaces according to their information gains. The standard deviation is also included in this table.

Dataset	10 ten-fold accuracy (%)	Number of rules	Number of useful attributes
australian	85.68 ± 0.31	5.69 ± 0.40	3.00 ± 0.00
breast-cancer	73.52 ± 0.60	5.51 ± 0.21	2.00 ± 0.00
breast-w	95.43 ± 0.32	6.30 ± 0.85	3.00 ± 0.00
german	72.47 ± 0.62	3.64 ± 0.33	2.00 ± 0.00
glass (G2)	82.82 ± 1.78	7.53 ± 0.62	3.00 ± 0.00
heart-c	82.44 ± 0.94	9.74 ± 0.37	3.98 ± 0.04
heart-h	79.03 ± 0.97	4.78 ± 1.03	2.11 ± 0.10
heart-statlog	81.15 ± 0.77	14.40 ± 0.18	4.72 ± 0.13
hepatitis	79.45 ± 2.14	3.37 ± 0.20	4.11 ± 0.10
horse-colic	82.10 ± 0.35	3.61 ± 0.21	2.00 ± 0.00
iris	94.80 ± 0.82	3.99 ± 0.37	1.51 ± 0.09
lymphography	80.38 ± 1.72	15.50 ± 1.03	5.00 ± 0.00
primary-tumor	40.92 ± 0.86	23.87 ± 0.32	5.00 ± 0.00
sick	97.61 ± 0.04	4.10 ± 0.11	6.00 ± 0.00

Table 4.2: The result of the GRG experiment

4.4 Comparisons

4.4.1 Comparisons to Other Machine Learning Approaches

We compared the GRG results with decision tree methods C5.0 (a successor to C4.5 method (Quinlan, 1993) and M5' (Frank, et al. 1997), and neuron networks approach N2C2S (Setiono, 2001). The detailed comparisons with ten 10-fold accuracy and standard deviation are listed in Table 4.3.

Dataset	GRG	C5.0	M5'	N2C2S
australian	85.68 ± 0.31	85.30 ± 0.50	85.80 ± 0.90	84.80 ± 0.70
breast-cancer	73.52 ± 0.60	73.30 ± 1.60	69.60 ± 2.30 •	67.80 ± 2.10 •
breast-w	95.43 ± 0.32	94.50 ± 0.30 •	95.30 ± 0.30	96.50 ± 0.20 ◊
german	72.47 ± 0.62	71.20 ± 1.00	72.90 ± 0.70	70.10 ± 1.60 •
glass (G2)	82.82 ± 1.78	78.70 ± 2.10 •	81.80 ± 2.20	77.90 ± 2.50 •
heart-c	82.44 ± 0.94	76.80 ± 1.40 •	80.90 ± 1.40	82.40 ± 1.90
heart-h	79.03 ± 0.97	79.80 ± 0.90	79.00 ± 0.80	81.60 ± 1.60 ◊
heart-statlog	81.15 ± 0.77	78.70 ± 1.40 •	82.20 ± 1.00	77.50 ± 1.00 •
hepatitis	79.45 ± 2.14	79.30 ± 1.20	81.90 ± 2.20	81.90 ± 3.30
horse-colic	82.10 ± 0.35	85.30 ± 0.60 ◊	84.60 ± 0.70 ◊	78.90 ± 1.20 •
iris	94.80 ± 0.82	94.50 ± 0.70	94.70 ± 0.70	96.60 ± 1.60 ◊
lymphography	80.38 ± 1.72	75.40 ± 2.80 •	79.80 ± 1.40	82.90 ± 1.90 ◊
primary-tumor	40.92 ± 0.86	41.80 ± 1.30	45.10 ± 1.60 ◊	45.90 ± 1.30 ◊
sick	97.61 ± 0.04	98.80 ± 0.10 ◊	98.30 ± 0.10 ◊	97.50 ± 0.10

Table 4.3: The comparison of the accuracy of GRG, C5.0, M5' and N2C2S

For comparing the accuracy of any two methods shown above, we conducted the t statistic testing. The t statistic for testing the null hypothesis that the two means are equal was obtained and a two-tailed test was conducted. At the significance level of $\alpha = 0.01$, if the null hypothesis was rejected, then the two methods were checked to see which one yielded the higher average accuracy rate. In Table 4.4, a bullet closed circle (•) is denoted for a GRG algorithm win, while a diamond is denoted for a GRG loss (◊). The summary of the comparisons are presented in Table 4.4.

GRG versus	Wins (•)	Ties	Losses (◊)
C5.0	5	7	2
M5'	1	10	3
N2C2S	5	4	5

Table 4.4: Summary of the comparison result of GRG, C5.0, M5' and N2C2S

As shown in the table, GRG algorithm performs better than C5.0 but loses to M5' in 2 more data sets, and tied with N2C2S. As summarized, GRG loses in those data sets which have more attributes or more classes (as sick and primary tumor etc). As the time complexity grows exponentially during the rule merging process in GRG algorithm, only a limited number of attributes were selected for the rule generation. If the selected

attributes have very high information gains, they would be sufficient for classification. However, if all attributes do not differ much in information gains, then the few selected ones are not capable to generate accurate rules accurate enough to cover all the data. In this case, the performance of GRG is not satisfactory. On the other hand, GRG performed well for those data sets with fewer classes and with fewer attributes or with attributes which have high information gains. The rule merging process, in this case, was not bounded by any limit, and therefore it could generate accurate rules and outperform other methods.

4.4.2 Comparisons to Other Rule List Generation Algorithms

We also compare GRG to some other methods which share the same rule list generation nature as GRG. Clark et al. (1989) selected three medical data sets for their experiment and comparison with older AQ15 (Michalski et al. 1986) and Bayes methods. So we also select the same three data sets here for comparison. The first of the three data sets concerns Lymphography. The data in Lymphography are consistent, i.e. examples having identical attribute values do not result in the same class. But this data set may contain errors as it was not submitted to a detailed checking after its original compilation by the Medical Centre. The second data set is the prognosis of Breast Cancer. It was verified after collection and thus is likely to be free of errors. The third data set is the location of Primary Tumor. Physicians distinguish between twenty-two possible tumor locations, thus the number of classes (twenty-two) is large in this data set. This data set is also relatively error free because of verification; however, it contains inconsistent data and missing attribute values.

The comparison of GRG and these rule list generation approaches in these three data sets is summarized in Table 4.5 and Table 4.6. The table shows interesting regularities. In the data sets of Breast Cancer and Primary Tumor, GRG has achieved significantly better accuracies compared to CN2 and AQ. The numbers of rules generated by GRG in these two data sets are also within the rule number ranges of CN2 and AQ. In the data set of Lymphography, GRG showed similar accuracy and number of rules in comparison to CN2 and AQ. The reason why GRG in this case has not outperformed the other

approaches is due to the relatively even distribution of information gain in each attribute. Since GRG has utilized only a small portion of the attributes, the information gains from these attributes are limited, and the advantages of the algorithm itself are thus not significant compared to the other two approaches.

Accuracy (%)	Lymphography	Breast Cancer	Primary Tumor
GRG	80	74	41
CN2 [*]	78-82	70-71	36-37
AQ15 [†]	80-82	66-68	29-41
Human Experts	85 [‡]	64	42

Table 4.5: The comparison of accuracies of GRG, CN2, AQ15 and human experts in three medical domains

	Lymphography		Breast Cancer		Primary Tumor	
	Selectors [§]	Rules	Selectors	Rules	Selectors	Rules
GRG	35	19	11	22	75	24
CN2 ^{**}	-	12-24	-	4-28	-	19-42
AQ15	10-37	76	7-160	208	112-551	562

Table 4.6: The comparison of complexities of GRG, CN2, AQ15 and human experts in three medical domains

We have also compared the training and testing accuracies of the three learning algorithms in the three data sets in medical domains. Table 4.7 shows the details of this comparison. The overfitting problems of the three learning algorithms can be studied from this table. For AQ15, because of the nature of this algorithm, which specializes complexes until they cover all the positive and negative examples, the accuracy of the training data is consequently very high. For Lymphography and Breast Cancer, the training accuracy of AQ15 is as high as 100%, and for Primary Tumor, its training accuracy is 75%, almost doubles the training accuracy of CN2 and GRG. On the other

* The accuracies and complexities varies because of the use of three different threshold in CN2: 90%, 95%, and 99%

† The accuracies and complexities varies because of the use of three different strategies in AQ15: no-truncation, eliminate-unique-cover and best-complex

‡ Estimated ()

§ Selector is defined in Section 2.2.1

** The number of attributes (selectors) is not stated in the original paper

hand, its testing accuracy is significantly lower than both GRG and CN2; therefore, the problem of overfitting in AQ15 is very prominent in these three data sets. For CN2, its training and testing accuracies are quite close, especially for Breast Cancer and Primary Tumor data sets; and it shows the noise data handling in CN2 has achieved good result and has prevented it from overfitting the training data. GRG has achieved similar results as CN2 in Lymphography. GRG's training accuracies are slightly higher than CN2 in Breast Cancer and Primary Tumor, but GRG also has significantly higher testing accuracies in these two data sets than CN2.

Accuracy (%)	Lymphography		Breast Cancer		Primary Tumor	
	Train	Test	Train	Test	Train	Test
GRG	90	80	77	74	50	41
CN2 (99% Threshold)	91	82	72	71	37	36
AQ15	100	76	100	72	75	35

Table 4.7: The comparison of the training and testing accuracy of GRG, CN2 and AQ15 in three medical domains

4.4.3 A Case Study of GRG

To illustrate the GRG approach, we follow an actual fold in rule generation case, the Breast Cancer data set.

After the random shuffling of examples in Breast Cancer data set, we divided the shuffled examples to ten equal sized portions (ten folds). Nine of them are used for training while the one left is used for testing. As there is no numeric attribute in Breast Cancer data set, Chi2 discretization is not conducted. For all the nine discrete attributes, information gain is calculated:

```
Information gain for attribute 0 is 0.01060595
Information gain for attribute 1 is 0.00200161
Information gain for attribute 2 is 0.05717112
Information gain for attribute 3 is 0.06899508
Information gain for attribute 4 is 0.05342264
Information gain for attribute 5 is 0.07700985
Information gain for attribute 6 is 0.00248898
Information gain for attribute 7 is 0.01506662
Information gain for attribute 8 is 0.02581902
```

These attributes are then sorted regarding to their respective value of information gain. The attributes with largest information gains are then selected to form the subspaces.

Selected attributes are 5, 3
Dimension of subspaces is 3x7

Since the user defined upper limit for subspaces size is 100 in this case, only two attributes (5 and 3) are selected to form subspaces of dimension 3x7, where 3 and 7 are the number of different values of attribute 5 and attribute 3 respectively (a missing value is treated as an attribute value, and attributes values that have zero corresponding examples are annexed to other values using disjunction), and the size of subspaces is then 3x7=21 which is under the upper limit 100. If another attribute with the third largest information gain is also selected in this case, which is attribute 2, the dimension of subspaces would be 3x7x11, where 11 is the number of unique values of attribute 2. The size of subspaces then would be 3x7x11=132, which would exceed the upper limit of 100. Therefore, only attribute 5 (deg-malig) and 3 (inv-nodes) are used to form the subspaces. The initial subspaces of this fold are shown in Table 4.8.

		inv-nodes						
		0-2	3-5	6-8	9-11	12-14	15-23	24-39
deg-malig	1	(54, 10) no-rec	(3, 0) no-rec	(0, 0) unlabeled	(1, 0) no-rec	(0, 0) unlabeled	(0, 0) unlabeled	(0, 0) unlabeled
	2	(72, 14) no-rec	(10, 7) no-rec	(6, 1) no-rec	(2, 2) no-rec ^{††}	(0, 0) unlabeled	(1, 0) no-rec	(0, 0) unlabeled
	3	(27, 13) no-rec	(3, 7) rec	(1, 9) rec	(2, 3) rec	(1, 2) rec	(2, 3) rec	(0, 1) rec

Table 4.8: The distribution of Breast Cancer samples according to their deg-malig and inv-nodes

The subspaces with the highest example frequency is chosen to form \bar{R} : if deg-malig = 2 and inv-nodes = (0-2), then the class is no-rec. Then rules that have the same class values \bar{R} as or “unlabeled” class value are found and added to a rule list:

```
Deg-malig=1 and inv-node=0 to 2 --> no-rec
Deg-malig=1 and inv-node=3 to 5 --> no-rec
Deg-malig=1 and inv-node=6 to 8 --> unlabeled
Deg-malig=1 and inv-node=12 to 14 --> unlabeled
```

^{††} When a draw occurs, the class label is chosen randomly from the classes with highest sample frequencies

```

Deg-malig=1 and inv-node=15 to 23 --> unlabeled
Deg-malig=1 and inv-node=24 to 39 --> unlabeled
Deg-malig=2 and inv-node=0 to 2 --> no-rec
Deg-malig=2 and inv-node=3 to 5 --> no-rec
Deg-malig=2 and inv-node=6 to 8 --> no-rec
Deg-malig=2 and inv-node=9 to 11 --> no-rec
Deg-malig=2 and inv-node=12 to 14 --> unlabeled
Deg-malig=2 and inv-node=15 to 23 --> no-rec
Deg-malig=2 and inv-node=24 to 39 --> unlabeled
Deg-malig=3 and inv-node=0 to 2 --> no-rec

```

These rules are merged to form new rules that cover more subspaces. Among all the rules (original and newly merged), those that contain \bar{R} and are not contained in other rules are selected as optimal rule candidates:

```

Rule #1: deg-malig=1 to 2 and inv-node=0 to 39 --> no-rec
Rule #2: deg-malig=1 to 3 and inv-node=0 to 2 --> no-rec

```

From the above two rules, one rule will be chosen to be included in the final rule list. We first compare the total number of samples belonging to the class of \bar{R} covered by the rules. Rule #1 covers 148 such samples and Rule #2 covers 153 such samples. Since Rule #2 has already contained more samples than Rule #1, the comparisons of number of irrelevant attributes and the number of subspaces covered will not be conducted. And Rule #2 is added to the final rule list. Then we reset the labels of the subspaces covered by Rule #2. Table 4.9 shows the updated subspaces.

		inv-nodes						
		0-2	3-5	6-8	9-11	12-14	15-23	24-39
deg-malig	1	(0, 0) unlabeled	(3, 0) no-rec	(0, 0) unlabeled	(1, 0) no-rec	(0, 0) unlabeled	(0, 0) unlabeled	(0, 0) unlabeled
	2	(0, 0) unlabeled	(10, 7) no-rec	(6, 1) no-rec	(2, 2) no-rec	(0, 0) unlabeled	(1, 0) no-rec	(0, 0) unlabeled
	3	(0, 0) unlabeled	(3, 7) rec	(1, 9) rec	(2, 3) rec	(1, 2) rec	(2, 3) rec	(0, 1) rec

Table 4.9: The updated distribution of Breast Cancer samples according to their deg-malig and inv-nodes after first iteration

Then we begin another round searching for best rule. The subspace with highest sample frequency this round is deg-malig=2 and inv-node=3 to 5 and \bar{R} is thus *if deg-malig=2 and inv-node=3 to 5 then the class is no-rec*. Like the previous iteration, we find the rules

that have the same class value as \bar{R} or “unlabeled” class value and merge them. Then rules containing \bar{R} and not contained in other rules are selected as candidates:

Rule #1: deg-malig=1 to 2 and inv-node=0 to 39 --> no-rec

Since there is only one such rule, no comparison is needed. And this rule is added to the final rule list. Table 4.10 shows the updated subspaces after second iteration.

		inv-nodes						
		0-2	3-5	6-8	9-11	12-14	15-23	24-39
deg-malig	1	(0, 0) unlabeled	(0, 0) unlabeled	(0, 0) unlabeled	(0, 0) unlabeled	(0, 0) unlabeled	(0, 0) unlabeled	(0, 0) unlabeled
	2	(0, 0) unlabeled	(0, 0) unlabeled	(0, 0) unlabeled	(0, 0) unlabeled	(0, 0) unlabeled	(0, 0) unlabeled	(0, 0) unlabeled
	3	(0, 0) unlabeled	(3, 7) rec	(1, 9) rec	(2, 3) rec	(1, 2) rec	(2, 3) rec	(0, 1) rec

Table 4.10: The updated distribution of Breast Cancer samples according to their deg-malig and inv-nodes after second iteration

Now in the third iteration, \bar{R} is if *deg-malig=3 and inv-node=6 to 8 then the class is rec.*

And candidate rules are:

Rule #1: deg-malig=1 to 3 and inv-node=0 to 39 --> rec

Since only one rule is the candidate, it is added to the final rule list. Now the updated subspaces are shown in Table 4.11.

		inv-nodes						
		0-2	3-5	6-8	9-11	12-14	15-23	24-39
deg-malig	1	(0, 0) unlabeled	(0, 0) unlabeled	(0, 0) unlabeled	(0, 0) unlabeled	(0, 0) unlabeled	(0, 0) unlabeled	(0, 0) unlabeled
	2	(0, 0) unlabeled	(0, 0) unlabeled	(0, 0) unlabeled	(0, 0) unlabeled	(0, 0) unlabeled	(0, 0) unlabeled	(0, 0) unlabeled
	3	(0, 0) unlabeled	(0, 0) unlabeled	(0, 0) unlabeled	(0, 0) unlabeled	(0, 0) unlabeled	(0, 0) unlabeled	(0, 0) unlabeled

Table 4.11: The updated distribution of Breast Cancer samples according to their deg-malig and inv-nodes after third iteration

Now the subspaces all cover zero samples, and the rule generation process terminates.

The final rule list is returned:

Rule #1: deg-malig=1 to 3 and inv-node=0 to 2 --> no-rec
 Rule #2: deg-malig=1 to 2 and inv-node=0 to 39 --> no-rec
 Rule #3: deg-malig=1 to 3 and inv-node=0 to 39 --> rec

This is an ordered rule list, where Rule #1 has the highest priority and Rule #3 has the lowest. In all data sets, the last rule in the final rule list typically cover the entire example space, and thus it is also a default rule, i.e. when no other rule applies to a new example, this rule will apply. Therefore, the rule list can be interpreted as:

```
if inv-node=0 to 2 then no-rec  
else if deg-malig=1 to 2 then no-rec  
else rec
```

For example, if we encounter a new Breast Cancer instance: deg-malig=2 and inv-node=15 to 23, we consult the rule list in order. The first rule is first consulted, but it does not apply to the instance because the attribute value inv-node=15 to 23 in the new instance does not fall in the range of the rule. Then the second rule is consulted. Since the attribute value of the new instance deg-malig=2 falls in range of the second rule, the second rule can be applied and the class value “no-rec” is predicted.

In this fold, the training and testing accuracies are 78.21% and 75.86% respectively.

Chapter 5

Conclusion

5.1 Summary

In this Honours Year Project, We have implemented the Greedy Rule Generation algorithm (Odajima et al., 2006). GRG is a sequential covering algorithm that generates an ordered rule list from discrete data. To overcome the common problem of high complexity in rule generation algorithms, we added an upper limit to the number of subspaces and thus kept the complexity under control. We have also re-implemented Chi2 discretization (Liu et al, 1995) and information gain measures for attribute discretization and selection. The size of subspaces constructed based on these discrete and selected attributes is controlled under the upper limit because of the elimination of number of attributes and intervals of each attribute.

We conducted intensive ten 10-fold testing on 14 real world data sets in the experiment. We gathered the results of accuracy, rule list size and number of attributes used and compared to other algorithms. GRG has shown competitive performance in comparison with M5' (Frank, et al. 1997) and N2C2S (Setiono, 2001) and better performance over rule generation algorithms as AQ (Michalski et al. 1986) and CN2 (Clark et al., 1989).

5.2 Future Work

There are limitations of our approach as well. Although the use of upper limit of subspaces size has restricted the complexity; however, it may also cause information loss when only a small portion of attributes were retained for rule generation. Therefore, the recommendation for the future work includes increasing the number of attributes and attributes intervals allowed in rule generation (increasing the upper limit of subspaces

size) while still maintaining low complexity. In more detail, the rule merging process may be optimized by selecting mergeable pairs of rules to merge instead of merging all the mergeable pairs. Since rule merging accounts for most of the calculation time and space, if the process can be really optimized, the complexity would decrease greatly and make the algorithm more efficient.

Bibliography

Blake, C.L., and Merz, C.J. (1998) UCI Repository of machine learning databases (<http://www.ics.uci.edu/~mllearn/MLRepository.html>). Irvine, CA: University of California, Department of Information and Computer Science.

Cestnik, G., Kononenko, I., and Bratko, I. (1987) Assistant-86: A knowledge -elicitation tool for sophisticated users. In *Progress in Machine Learning*, I. Bratko and N. Lavrac, Eds. Wilmslow, U.K.: Sigma, pp. 31--45.

Clark, P., and Niblett, R. (1989) The CN2 induction algorithm. *Machine Learning*, 3, pp261-284.

Frank, E., Wang, Y., Inglis, S., Holmes, G., and I.H. Witten. (1998) Using model trees for classification. *Machine Learning*, 32:63--76, 1998.

Holmes, G., Donkin, A., Witten, I.H. (1994) WEKA: a machine learning workbench. In *Proceedings Second Australia and New Zealand Conference on Intelligent Information Systems*, Brisbane, Australia 1994, pp357-361

Kalbfleish, J., (1979) *Probability and statistical inference* (Vol. 2). Springer-Verlag, New York.

Liu, H., and Setiono, R. (1995) Chi2: Feature selection and discretization of numeric attributes. In *the Proceedings of the 7th International Conference on Tools for Artificial Intelligence*, pp388-391.

Liu, H., and Setiono, R. (1996) Dimensionality reduction via discretization. *Knowledge Based Systems* 9(1), pp67-72.

Liu, H., and Tan, S.T. (1995) X2R: A fast rule generator. In *the Proceedings of the 7th IEEE International Conference on Systems, Man and Cybernetics*, pp631-635.

- Michalski, R.S. (1969) on the quasi-minimal solution of the general covering problem. In the Proceedings of the Fifth International Symposium on Information Processing, pp125-128. Bled, Yugoslavia.
- Michalski, R.S., and Larson, J. (1983) The multi-purpose incremental learning system AQ15 and its testing application to three medical domains. In the Proceedings of the Fifth National Conference on Artificial Intelligence, pp1041-1045. Philadelphia: Morgan Kaufmann.
- Michalski, R.S., Mozetic, I., Hong, J., Lavrac, N. (1986) The AQ15 Inductive learning System: An Overview and Experiments. In Proceedings of the International Meeting on Advances in Learning, IMAL 1986.
- Odajima, K., Hayashi, Y., and Setiono, R. (200?) Greedy rule generation from discrete data and its use in neural network rule extraction. ???
- Pazzani, M., Brunk, C., and Silverstein, G. (1991) A knowledge-intensive approach to learning relational concepts. In the Proceedings of the 8th International Workshop on Machine Learning, pp432-436.
- Quinlan, J., R. (1983) Learning efficient classification procedures and their application to chess end games. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, Machine Learning: An Artificial Intelligence Approach, pp463--482. Tioga, Palo Alto, 1983.
- Quinlan, J., R. (1990) Learning logical definitions from relations. Machine Learning, 5, pp239-266.
- Quinlan, R. (1993) C4.5: Programs for machine learning. Morgan Kaufman, San Mateo, CA, 1993.
- Quinlan, R. (1998) C5.0: An Informal Tutorial. RuleQuest. <http://www.rulequest.com/see5-unix.html>.
- Setiono, R. (2001) Feedforward neural network construction using cross-validation. Neural Computation, 13(12):2865–2877, 2001.