Discovery of Classification Rules

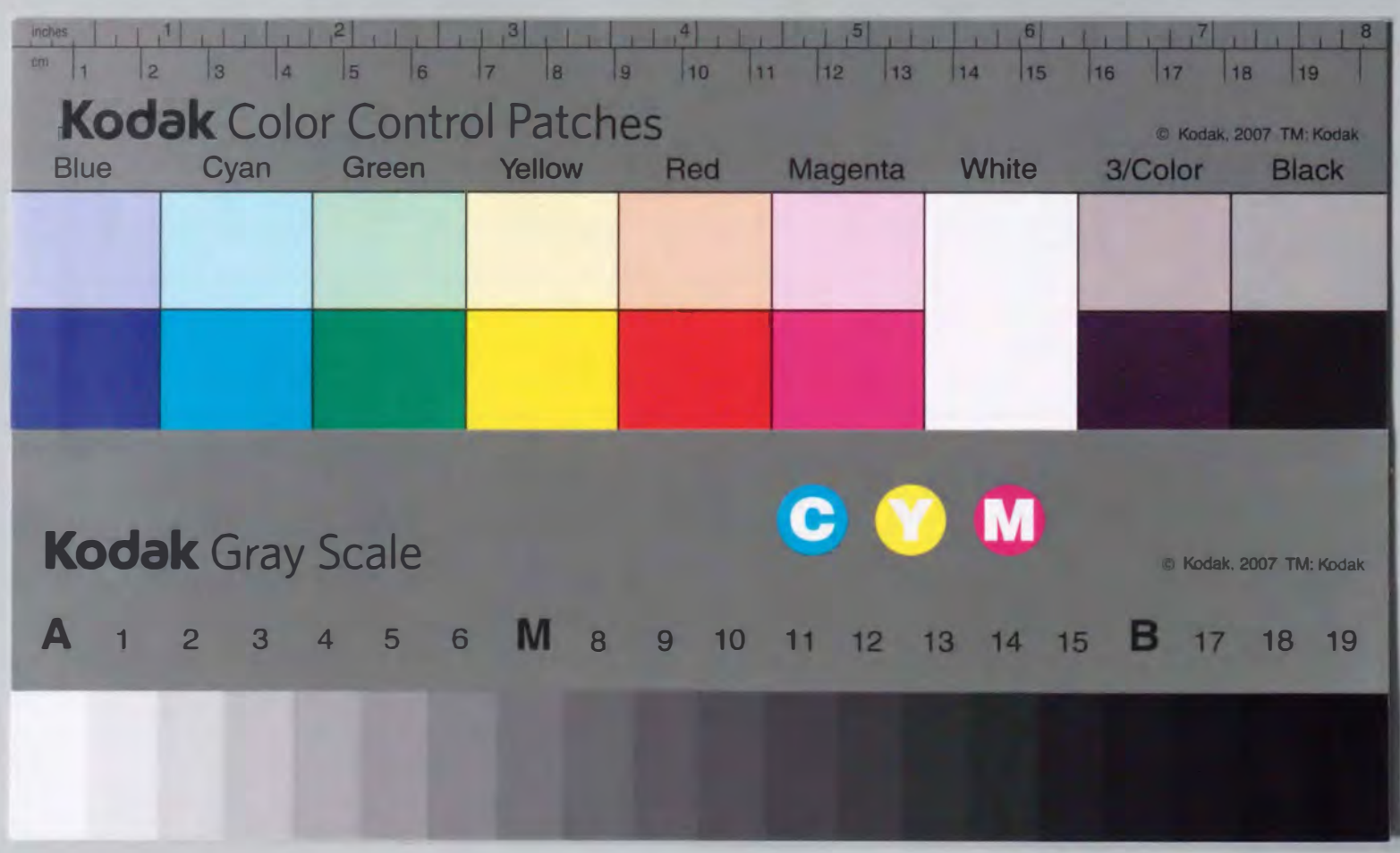
湯上, 伸弘
九州大学システム情報科学研究科情報理学専攻

https://doi.org/10.11501/3180411

出版情報：九州大学, 2000, 博士（理学）, 課程博士
バージョン：
権利関係：

Discovery of Classification Rules

Nobuhiro Yugami



(1)

Discovery of Classification Rules

Nobuhiro Yugami

February 2001

Abstract

This thesis investigates discovery of classification rules from databases for classifying new data and for analyzing properties of the databases. Typical databases involve huge number of rules and the key issues of rule discovery are how to evaluate interestingness of rules and how to find interesting rules efficiently.

In classification task, the desired rules are the ones that are as simple as possible and can classify new instances correctly. Recent researches on supervised learning show learning plural rule sets can significantly improve classification ability while it degrades the simplicity of the total rule sets. This thesis presents a discovery algorithm IGR(Instance Guided Rule induction) that achieves high classification ability without degrading simplicity by inducing one pseudo-optimum rule for each training instance.

In analysis of the properties of the databases, most previous researches use support and accuracy for selecting rules and tend to extract many statistically trivial rules. To resolve this difficulty, this thesis presents new criteria of rules that evaluate a rule by whether every condition in its body is really required to imply its conclusion. Moreover, this thesis presents two algorithms to extract interesting rules based on the criteria, DIG(Discovery of Interesting rules with Grouping attribute values) for nominal databases and IIS(Iterative Interval Specialization) for numeric databases.

The contributions of this thesis are summarized as follows. First, this thesis clarifies what kind of rules should be discovered and gives new criteria of rules to filter out less interesting rules. Second, this thesis presents three efficient discovery algorithms for classifying new instances and for analyzing properties of databases. Finally, this thesis empirically demonstrates the performance of the algorithms.

Acknowledgements

I would like to thank Professor Setsuo Arikawa of Kyushu University for supervising this thesis. He guided my research and gave many important and helpful advice for this thesis.

I am deeply grateful to researchers in Fujitsu Laboratories for thoughtful comments and valuable discussions, especially to Seishi Okamoto, Hiroya Inakoshi and Yuiko Ohta. Many results in this thesis are joint works with them.

I indebted to members of the seminar at department of informatics of Kyushu University for their helpful comments and advice, especially to Hiroki Arimura, Hiroki Ishizaka, Ayumi Shinohara, Takashi Shinohara, Hiroshi Sakamoto and Masayuki Takeda.

I also grateful to Hiromu Hayashi, Kazuhiro Matsuo, Fumihiro Maruyama, Ryusuke Masuoka, Hirotaka Hara for their strong support to my research at Fujitsu Laboratories and for giving a chance to research at Kyushu University.

Finally, all love and gratitude to my wife, Masayo Yugami, for her devotion and invaluable assistance during this research.

Contents

Chapter 1

Introduction

A key element of an intelligent system is the ability to discover new knowledge. This ability has been researched in both of machine learning and data mining. Knowledge discovery is a complex task consists of many kinds of processes, gathering various kinds of data from distributed resources, cleaning gathered data by editing them (semi-) automatically, combining the data to generate a database to be analyzed, analyzing the database, and presenting the results of analysis to a user[8, 24, 39, 57, 67]. Even if all of these processes play quite important roles in knowledge discovery, the central issue of knowledge discovery is the process of data analysis. Many paradigms have been proposed for data analysis[11, 36] depending on what kind of knowledge is discovered from what kind of data and depending on how the discovered rules are used for. This thesis focuses on discovery of classification rules from labeled instances for predicting classes of new instances accurately and for extracting non-trivial regularities in the given instances.

1.1 Rule Discovery for Prediction

Many applications of knowledge discovery are prediction tasks from prior instances[67]. It is an active subfield of machine learning to learn hypotheses from a set of labeled instances to classify new instances. Many learning paradigms have been proposed to

Table 1.1: An example of training instances. In this example, each instance is represented by its class label and values for three attributes, *Made_in*, *Size* and *Price*.

ID	class	*Made_in*	*Size*	*Price*
1	positive	*japan*	*small*	1,000,000
2	positive	*japan*	*small*	1,500,000
3	positive	*japan*	*small*	2,500,000
4	positive	*japan*	*middle*	1,200,000
5	positive	*japan*	*middle*	3,000,000
6	negative	*japan*	*big*	1,800,000
7	positive	*europe*	*small*	1,900,000
8	negative	*europe*	*small*	2,800,000
9	negative	*europe*	*middle*	1,000,000
10	negative	*europe*	*middle*	3,500,000
11	negative	*europe*	*big*	1,400,000
12	positive	*europe*	*big*	2,000,000

learn various types of hypotheses from various types of instances[41]. As the representation of the instances, this thesis assumes attribute-value representation in which each instance is represented by its class label and values of predefined attributes. This representation is simple but sufficiently powerful in many practical domains and is used in many learning systems to learn decision trees[51, 10], decision rules[38, 15, 17], neural networks[31], and nearest neighbor classifiers[19, 47]. Table 1.1 is an example of the representation of training instances. In this table, each row represents one instance and each column shows the values for one attribute.

This thesis focuses on discovery of decision rules, a set of classification rules, from the training instances with attribute-value representation. A classification rule is a kind of if-then rule that constrains values of attributes in its body and classifies the instances into the class in its conclusion. Following is an example of classification rule learned from the training instances in table 1.1.

$$Size \in \{small, middle\} \wedge Price \leq 1,800,000 \Rightarrow class = positive.$$

This rule means that an instance belongs to class *positive* if it satisfies the two conditions on *Size* and *Price*. This thesis is only concerned with classification rules whose bodies are conjunctions of conditions that constrain values of single attributes.

In rule discovery for prediction, a hypothesis is a set of discovered rules and a new instance is classified with the whole set of discovered rules. Then, discovered rules are evaluated as a set of the rules.

Learning a hypothesis from a set of training instances is a kind of optimization problem because learning is a task to select the best hypothesis with respect to a certain criterion among huge number of hypotheses[43]. This optimization problem is quite difficult because of the following two reasons. First difficulty is computational intractability. If there are M attributes with D possible values, then the number of distinct instances is D^M and the number of logically distinct hypotheses becomes 2^{D^M} for each class because a hypothesis is defined by a set of instances covered by it. Second difficulty is how to evaluate hypotheses. In many application of machine learning techniques, the primary objective of the learned hypothesis is to classify new, unknown instances correctly and the best hypothesis is the one with the highest classification accuracy. However, learning algorithms only know the given training instances and can not evaluate exact accuracy for new instances. They can only predict the accuracy for new instances from the accuracy for known instances[4]. Strict optimization with such an uncertain evaluation causes over-fitting with the training instances[46, 52].

Most learning algorithms do not try strict optimization because of the above two difficulties. They apply more computationally light methods for the main process of learning and try optimization in pre-processes or post-processes of learning such as feature subset selection[12, 35] and pruning decision trees[18, 42] and decision rules[16]. Some researches tried strict optimization but were effective for only a few databases[29,

46, 70].

In addition to classification performance, simplicity of learned hypotheses is another viewpoint of hypotheses evaluation. Simplicity becomes important when human experts try to understand learned hypotheses and to modify them based on their domain knowledge. In such a case, simple hypotheses with small number of rules are easy to understand and are more preferable than complex hypotheses.

It is well known that if two hypotheses have similar classification ability on the training instances, then the simpler one works better for unknown instances[7]. This heuristic is known as "Occam's razor". Most learning algorithms use Occam's razor explicitly or implicitly. For example, Quinlan and Rivest [54] apply minimum description length (MDL) principle[55] to decision tree induction that minimizes the total description length of a tree itself and descriptions for exceptional training instances miss-classified by the tree. However, even if simpler hypotheses usually achieve high accuracy, these two criteria evaluate different properties of hypotheses and don't need to coincide with each other.

Recent researches show counter examples of Occam's razor. For example, bagging[9] and boosting[58] learn plural hypotheses with weak learners and combine them as a whole hypothesis to classify new instances. Then, the whole hypothesis is more complex than individual hypotheses. However, both of empirical[25, 53] and theoretical analyses[59] show their abilities to improve classification accuracy of the existing learning algorithms. A simple hypothesis is more preferable than complex ones because of its simplicity itself, not because it has higher classification ability[21].

1.2 Rule Discovery for Extracting New Knowledge

The purpose of rule discovery is sometimes not to predict for new instances but to extract interesting regularities to analyze the characteristics of the given databases. Association rules discovery[1] is an example of this type of rule discovery.

In discovery of association rules, each training instance is a set of items and the purpose is to analyze relationships between items. Basket analysis is a typical application of association rule discovery. In this analysis, an instance is a set of items bought by one person and the purpose of the analysis is to generate rules such as "if a customer buys beer then he/she also buys potato chips with high probability".

As in rule discovery for prediction, evaluation of rules is an important issue in discovery for analysis. However, there is a big difference between these two kinds of discovery on what should be evaluated. In prediction tasks, the purpose of discovery is to discover a set of rules, not a specific rule, to achieve high prediction performance and the discovered rules are evaluated as a set of them. Instead, in discovery for analysis, the purpose is to discover rules representing useful regularities and an individual rule is evaluated as itself.

In usual, discovery for analyzing databases is a kind of constraint satisfaction problems[65, 69, 72] to enumerate all rules that satisfy given constraints such as the maximum length of rule's body and thresholds on given criteria to evaluate rules. In association rules mining, most algorithms constrain rules to be discovered with lower bounds of their support and accuracy. However, it is not easy to set appropriate thresholds on support and accuracy. High thresholds lead only trivial rules that human experts already know and low thresholds lead too many rules. In addition, even if large support and high accuracy are important conditions of good rules, they are not sufficient conditions for good rules. Then, it is quite difficult to select useful rules

with only constraints on them and it is required to restrict rules with other criteria that represent usefulness or interestingness of rules directly[5, 37, 49].

1.3 Objective of This Thesis

The primary objective of this thesis is to construct practical discovery systems for classification rules. This objective consists of two problems. First problem is to clarify what kind of classification rules should be discovered. The answer to this question strongly depends on what the discovered rules used for. This thesis deals with two types of rule discovery, discovery for classifying new instances accurately, and discovery for extracting non-trivial regularities. The evaluation of rules is quite important in the second situation and this thesis proposes new criteria of rules for the situation.

Second problem is how to discover the desired classification rules effectively. In usual, there are huge number of possible rules even in a small database and the crucial problem of rule discovery is how to search desired rules in practical time. This thesis proposes effective discovery algorithms for many situations, discovery of rules to achieve high classification performance, discovery of interesting rules in nominal domains, and discovery of interesting rules in numeric domains.

1.4 Overview of This Thesis

This section provides the content of the remainder of this thesis.

Chapter 2 presents a rule discovery algorithm IGR(Instance Guided Rule induction) for classifying new instances correctly. The objective of this algorithm is to achieve high classification accuracy without increasing both in learning time and the number of learned rules. IGR achieves these requirements by learning one pseudo-optimum rule for each training instance. The pseudo-optimum rule means the rule with 100%

accuracy that covers as many positive instances as possible. The rule represents the widest area of the concluding class around the corresponding training instance. If a test instance to be classified is sufficiently near to a certain training instance, we can expect that the rule for the training instance is also pseudo-optimum for the test instance and the rule classifies the test instance correctly. This chapter also reports the experimental evaluation using databases from UCI machine learning repository[6] and compares IGR with other learning algorithms with respect to classification performance, learning speed, and the complexity of learned hypotheses.

Chapter 3 and 4 describe rule discovery for extracting non-trivial regularities from a given database and the purpose of discovered rules is not for classifying new instances but to analyze characteristics of the given database. As described above, huge number of rules are embedded in typical databases and the most important problem of rule discovery is how to filter out less interesting rules. This problem is not so serious in discovery of rules to classify new instances because discovery algorithms can reject rules that don't contribute to improve classification performance. However, even if a rule overlaps with other rules and doesn't improve classification accuracy at all, it may represents a different regularity from other rules by constraining values of different attributes and may be worth to be discovered for analyzing databases.

Chapter 3 discusses how to evaluate rules and presents new criteria of classification rules that requires a rule to be discovered is not a trivial conclusion from simpler rules. Chapter 3 also applies the criteria to rule discovery in nominal domains and presents a new rule discovery algorithm, DIG(Discover Interesting rules with Grouping attribute values), to discover rules with grouping attribute values[22, 44], i.e. rules that permit plural values for each attribute in their bodies. Attribute value grouping increases the representation power of rules and makes it easy to understand the meaning of

discovered rules.

Chapter 4 applies the criteria presented in Chapter 3 to numeric domains in which values of all attributes are numeric. This chapter focuses on a classification rule in which each condition in its body requires that a value of a certain attribute belongs to a specific interval. Because a slight difference of boundaries of the intervals does not affect the meaning of a discovered rule, this chapter assumes values of each attribute are discretized into dozens of values and tries rule enumeration from the discretized instances. Even if the discretization decreases the number of possible rules, one of the most important problems is also how to reduce the time complexity for rule enumeration. This chapter presents two discovery algorithms to enumerate rules in numeric domains and compares them empirically.

Chapter 5 surveys the literatures about knowledge discovery algorithms and discusses the relationships between them and the criteria of rules and the algorithms presented in this thesis.

Chapter 6 summarizes the contributions of this thesis and discusses future work.

Chapter 2

Rule Discovery for Classification

2.1 Learning Hypotheses for Classification

This chapter discusses learning hypotheses from pre-classified training instances to classify new instances correctly. This problem has been researched as supervised learning[11, 36] and many algorithms have been proposed. This thesis focuses on discovery of if-then rules, where a hypothesis to classify new instances is a set of all discovered rules. In supervised learning, the primary objective is to classify new instances correctly and the most important criterion of learned hypotheses is classification accuracy for new instances. In addition, simplicity of the learned hypotheses is also an important criterion of the hypotheses. This becomes crucial when human experts want to understand the meaning of learned hypotheses and to modify the hypotheses with their background knowledge.

These two criteria, classification performance and simplicity, don't coincide with each other. For example, bagging[9] and boosting[58] learn plural hypotheses and combine all of them to classify new instances. Both of empirical[25, 53] and theoretical[59] analyses show that bagging and boosting can significantly improve classification performance. However, they degrade simplicity of the total complexity of the hypotheses. The objective of this chapter is to achieve the same classification performance with

Table 2.1: An examples of training instances. Values for *Price* are discretized into three values, *cheap*, *middle* and *expensive*.

ID	class	*Made_in*	*Size*	*Price*
1	positive	*japan*	*small*	*cheap*
2	positive	*japan*	*small*	*middle*
3	positive	*japan*	*small*	*expensive*
4	positive	*japan*	*middle*	*cheap*
5	positive	*japan*	*middle*	*expensive*
6	negative	*japan*	*big*	*middle*
7	positive	*europe*	*small*	*middle*
8	negative	*europe*	*small*	*expensive*
9	negative	*europe*	*middle*	*cheap*
10	negative	*europe*	*middle*	*expensive*
11	negative	*europe*	*big*	*cheap*
12	positive	*europe*	*big*	*middle*

bagging and boosting without loss of the simplicity of the discovered hypotheses.

2.2 Discovery Algorithm:IGR

2.2.1 Overview of IGR

This section presents a new discovery algorithm IGR(Instance Guided Rule induction) to learn a set of if-then rules to classify new instances correctly. The input of this algorithm is a set of training instances represented by class labels and values of predefined attributes. IGR can only deal with nominal attributes and requires values of numeric attributes to be discretized before learning. Table 2.1 shows an example of a set of training instances. The instances in this table are the same instances in Table 1.1 in Chapter 1 but the values of the last attribute, *Price*, are discretized into three values, *cheap*, *middle* and *expensive*. The output is a set of if-then rules with weights for majority voting such as

$$Made\_in = japan \wedge Size \in \{small, middle\} \Rightarrow class = positive : 3.0,$$

$$Size = small \wedge Price \in \{cheap, middle\} \Rightarrow class = positive : 1.5,$$

$$Price \in \{cheap, expensive\} \Rightarrow class = negative : 4.0,$$

When classifying a new instance, IGR checks whether the new instance satisfies a body of each rule and returns the class that maximizes the sum of the weights of satisfied rules. For example, the instance with $Made\_in = japan$, $Size = small$ and $Price = cheap$ satisfies bodies of all of the above three rules. Then, the score for class *positive* becomes $3.0 + 1.5 = 4.5$ and the score for *negative* is 4.0. As a result, the instance is classified into *positive*.

As discussed in the previous section, the objective of this algorithm is to achieve high classification performance without increasing the number of learned rules. The basic idea of IGR is to learn the optimal rule for each training instance. The optimal rule for a instance i belonging to class c is the rule that covers as many instances in class c including i as possible and rejects all instances in other classes. The optimal rule represents the widest area of class c around the instance i and we can expect that the rule can classify new instances near i correctly. Because it is computationally difficult to induce the strict optimal rules, IGR induces a pseudo-optimal rule for each training instance by greedy search and achieves high classification performance by combining such locally pseudo-optimal rules.

When inducing a rule for a target instance i, IGR applies general-to-special greedy search. It starts from a rule with no condition and iteratively specializes it by adding a new condition with the highest information gain among all of possible conditions that accept the target instance. IGR doesn't try backtracking and never removes the selected condition. IGR stops specialization when the rule covers only positive instances. Positive instances are the training instances belong to the same class of the

target instance and negative instances are the training instances in other classes.

The trivial method to learn one rule for each instance is to learn each rule one by one. However, IGR specializes rules for many target instances at once to accelerate learning by sharing condition evaluation processes to select the best conditions to specialize the rules. For example, when selecting the first condition of the rules, every condition has the same score for the rules for all target instances in the same class and IGR doesn't need to iterate evaluation of the conditions for the first specialization for each rule. If the rules for plural target instances select the same condition at the first specialization, then the rules can share the condition evaluation process to select the second conditions for specialization. The complexity of condition evaluation process is the number of conditions times the number of instances covered by the current body because the process requires to count a frequency of instances for each combination of an attribute value and a class label in the instances that satisfy the current body. This process must be done for every condition for every rule and is the most time consuming process in learning classification rules.

2.2.2 IGR

Figure 2.1 shows a pseudo-code of IGR that is a recursive procedure to learn pseudo-optimum rules for a certain class. The inputs of the algorithm are a set of target instances for which pseudo optimal rules should be induced, $Targets$, a current body of the rules for the target instances in $Targets$, $Body$, and the weights of intances to avoid to induce many similar rules. The output of the algorithm is a set of pseudo-optimum rules for the training instances in $Targets$, $Rules$. IGR is first called with all positive instances in $Targets$ and with no condition in $Body$ ($Body = true$) to induce rules for all positive instances belong to the target class c. The weights of instances are initialized to 1 for all instances. IGR iteratively specializes the body $Body$ by adding an

procedure IGR($Targets, Body, Weight$)
  inputs:
    $Targets$: set of target instances belong to a target class c.
    $Weight$: weights of training instances.
    $Body$: current body (conjunction of conditions) for target instances.
  output:
    $Rules$: set of discovered rules.
1. $Rules := \phi$.
   Pos:= set of positive instances that satisfy $Body$ and belong to a target class c.
   Neg:= set of negative instances that satisfy $Body$ and belong to a target class c.
2. **If** $Neg = \phi$ **then**
3. $Rules := \{Body \Rightarrow c\}$.
4. **else**
5. Generate all conditions to specialize $Body$ and evaluate them with
    weighted information gain in the set of instances $Pos \cup Neg$.
6. **Foreach** target instance $i \in Targets$
7. Select a condition that accepts t and maximizes information gain.
8. $I(t) := \{i | \text{condition } t \text{ is selected for } i\}$.
9. **Foreach** t such that $I(t) \neq \phi$ (decreasing order of information gain)
10. $Rules := Rules \cup IGR(I(t), Body \wedge t, Weight)$.
11. **Foreach** $p \in Pos$
12. $Weight[p] := \alpha \cdot Weight[p]$. $(\alpha \leq 1)$
13.**Return** $Rules$.

Figure 2.1: Discovery algorithm IGR. IGR learns rules for the training instances in $Targets$ and IGR is first called with all positive instances in $Targets$.

appropriate condition until $Body$ covers only positive instances (step 3). If $Body$ rejects all negative instances, IGR returns a rule $Body \Rightarrow c$. Otherwise, IGR evaluates all of possible conditions to specialize $Body$ and selects the best condition for each target instacne in $Targets$. IGR uses weighted information gain for evaluating conditions. The next subsection will explain this criterion for condition selection. For all target instances that select the same condition t, $I(t)$, IGR can evaluate the conditions for the next specialization at once and calls IGR recursively with $Targets = I(t)$. IGR updates the weights of intances to avoid to induce many similar rules. IGR decreases

the weights of intances accepted by the selected condition and focuses on the instances rejected by the condition in the discovery process with other conditions(step 10∼12).

The number of rules induced by IGR is at most the number of training instances and is usually smaller than that because plural training instances share one rule as their psuedo-optimal rule. This happens when the condition in step 2 in Figure 2.1 is satisfied with plural target instances in $Targets$.

IGR is similar to AQ family that also learns one rule for one seed instance. The most important difference between them is that IGR induces redundant rules to classify training instances correctly. After learning one rule, AQ removes all positive instnaces covered by the rule and induces the next rule for the remaining instances. The resulting rule sets cover all of positive instances and can classify all training instances correctly. Instead, IGR does not remove instances covered by the generated rules and induces one rule for every training instnaces. As a result, IGR learns more rules than AQ and some of the rules from IGR may not be required to explain training instances correctly. However, these redundant rules improve classification performance for new, unknown instances as the experimental results in section 2.3 will show. Another differenece is that AQ induces rules one by one but IGR induces plural rules at once to reduce running time.

2.2.3 Condition Selection for Specialization

For general-to-special rule induction like IGR, one of key issues of the algorithm is how to select conditions for specializing bodies of rules. When specializing a rule, the simplest and the most traditional way is to add a condition that accepts instances with a certain attribute value. However, this type of condition tends to generate too specific rules when each attribute has many possible values. To avoid over specialization, IGR divides values of one attribute into two groups and uses a condition that accepts values

in one group. When inducing rules for class c, IGR divides values based on whether each value increases the probability of the target class c or not as follows.

$$G^+(x, c, Body) = \{v | v \in Domain(x) \wedge P(c|x = v \wedge Body) > P(c|Body)\},$$
$$G^-(x, c, Body) = \{v | v \in Domain(x) \wedge P(c|x = v \wedge Body) \leq P(c|Body)\},$$

where x and $Domain(x)$ is an attribute and a set of its possible values. Therefore, IGR generates two conditions, $x \in G^+(x, c, Body)$ and $x \in G^-(x, c, Body)$, as candidates for specialization for each attribute x at each specialization step. For simplicity, I use $x = v$ to denote $x \in G^+(x, c, Body)$ ($x \in G^-(x, c, Body)$) if $G^+(G^-)$ involves only one value v. IGR fixes the value groups at each specialization step and doesn't modify the combination of the permitted values for the selected conditions afterwards. Instead, IGR may change the permitted range of the attribute by adding a new condition on the values of the same attribute.

IGR uses weights of instances to avoid inducing many similar rules and counts probabilities by replacing frequencies of instances with the sums of the weights of the instances. For example, $P(c|Body)$ becomes the ratio of the sum of the weights of the instances that satisfy both of $class = c$ and $Body$, on the sum of the weights of the instances that satisfy $Body$. IGR updates the weights of training instances with a dumping factor α at step 12.

IGR evaluates each condition with information gain[51] with small modification. Information gain of a condition t is defined as

$$\begin{aligned} IG(t, c, Body) &= E(P(c|Body)) \\ &- P(t \wedge Body) \cdot E(P(c|t \wedge Body)) \\ &- P(\neg t \wedge Body) \cdot E(P(c|\neg t \wedge Body)), \end{aligned} \tag{2.1}$$

where

$$E(p) = -p \cdot \log_2 p - (1 - p) \cdot \log_2(1 - p).$$

The probabilities in (2.1) are also counted with weighted frequencies of the instances. Information gain becomes maximum if the condition can completely divide instances into the set of positive instances and the set of negative instances. Oppositely, information gain becomes zero when the condition does not change the probability of the target class at all.

Information gain evaluates a division of instances into two sets and gives the same score to a condition t and its negation $\neg t$. In other words, it can't distinguish t and $\neg t$ and can't judge which is better for specialization. It is not important in decision tree learning because sub trees grow for both accepted and rejected instances in decision trees. However, decision rule learner focuses on only accepted instances and requires to distinguish the conditions and their negations. For example, if t permits most positive and few negative instances and gets high information gain, then $\neg t$ rejects most of positive instances and remains many negative instances. In such a case, it is good to specialize a current body with t but is stupid to use $\neg t$. Therefore, IGR checks whether the condition increases the probability of target class c and uses negative of information gain as a score of the condition if the probability deceases. As a result, IGR uses the following evaluation function for a condition t to specialize a body $Body$ when inducing rules for a class c.

$$Est(t, c, Body) = \begin{cases} IG(t, c, Body) & \text{if } P(c|t \wedge Body) \geq P(c|Body) \\ -IG(t, c, Body) & \text{otherwise} \end{cases} \tag{2.2}$$

2.2.4 Weights of Rules

IGR learns a set of if-then rules and classifies new instances with the rules. There are two common problems to classify a new instance by a set of rules, how to classify

an instance that satisfies no rule and how to classify an instance that satisfies plural rules. In the former case, IGR returns the majority class in the training set. In the later case, IGR selects a class by majority voting with the following weights of rules.

$$weight(r) = \sum_{i \in I(r)} \frac{1}{|S(i)|}, \tag{2.3}$$

where $I(r)$ is the set of training instances covered by a rule r and $S(i)$ is the set of rules that cover an instance i. This definition of weights assigns large weight to an isolated rule that covers many training instances not covered by other rules. Oppositely, even if a rule has large support and covers many instances, its weight may become small if it covers only instances covered by many other rules.

From a statistical viewpoint, a rule becomes more reliable as it covers more positive instances and less negative instances. Because IGR generates rules that cover only positive instances, the simplest weights of rules are their support, the number of positive instances covered by them. The recent version of CN2[14] sets weights of rules based on this approach. When classifying a new instance, it sums up class distributions of rules satisfied by the instance to be classified and classifies the instance into the class with maximum score. Because IGR induces rules covering only positive instances, using the support of rules as their weight coincide with the rule weighting in CN2. If the two rules with different class labels cover an instance to be classified, then it is natural to classify the instance into the class of the rule with larger support. However, using support as weights of rules causes a problem when the instance is covered by plural rules for each class. Intuitively, when plural rules with one class covers the instance to be classified, classification of the instance into the class is supported by the training instances covered by at least one of the matched rules. Then, if the learned rules don't overlap at all and each training instance is covered by only one rule, it is adequate to use the sum of the support of the rules as the score of the class. However, there may

be many rules that cover quite similar sets of the training instances and conclude the same class. If many similar rules cover an instance to be classified, the sum of their support becomes much larger than the number of training instances covered by them and the score becomes too high. In such a case, the training instances covered by many rules affect the results of classification of new instances much stronger than the instances covered by a few rules. A solution to this problem is to keep a set of covered training instances for each rule and to select the class that maximizes the size of the union of the instance sets of the matched rules for the class. However, this makes classification too slow when training sets are huge. Instead of calculating the union, IGR first counts how many rules cover each training instance and assigns a weight to a rule by summing the inverse of the number of rules for each training instance covered by the rule as in equation (2.3). This definition of the weight directly leads the following property.

$$\forall i, \sum_{r \in S(i)} weight(r) = 1.$$

This property means that every training instance gives the same contribution to the weights of rules.

2.2.5 Example

This subsection describes how IGR learns classification rules by using the training set shown in Table 2.1 as an example. To discover rules for a class positive, IGR is called with a set of all positive instances $\{1, 2, 3, 4, 5, 7, 12\}$ as $Targets$ and $Body = true$. Then, Pos and Neg in step 1 in Figure 2.1 are the sets of all positive and all negative instances respectively. The weights of instances are set to 1 for all instances. Because Neg includes four negative instances and doesn't satisfy the condition in step 2, IGR counts class distribution for each attribute value and generates possible conditions by

grouping attribute values (step 5). For example, class distributions for values of $Size$ are as follows.

	positive	negative
$Size = small$	4	1
$Size = middle$	2	2
$Size = big$	1	2
total	6	4

IGR divides values of each attribute into two sets by whether the probability of positive instances increases or not. Because only $Size = small$ increases the probability of positive instances and other two values decreases the probability, IGR generates two conditions for the attribute, $Size = small$ and $Size \in \{middle, big\}$. Information gain is 0.10 for dividing instances in Pos and Neg by whether $Size = small$ or $Size \in \{middle, big\}$,and the scores for these two conditions become 0.10 and -0.10 respectively. IGR also generates conditions on other two attributes. As a result, IGR generates the following six conditions for specialization.

condition	score
$Made\_in = japan$	0.20
$Size = small$	0.10
$Price = middle$	0.04
$Size \in \{middle, big\}$	-0.04
$Price \in \{cheap, expensive\}$	-0.10
$Made\_in = europe$	-0.20

For each target instance in $Targets$, IGR selects the condition with the highest score among the conditions that accept the target instance (step 7). The condition $Made\_in = japan$ has the highest score and IGR selects this condition for the target instances that satisfy it, i.e. for instances 1, 2, 3, 4 and 5. Target instances 7 and 12 don't satisfy this condition and IGR selects $Size = small$ for the instance 7 and $Price = middle$ for the instance 12. Then, IGR calls IGR recursively for these three conditions in descending

order of their scores (step 9 and 10).

IGR first selects the condition $Made\_in = japan$ and calls IGR with $Targets = \{1, 2, 3, 4, 5\}$ and $Body = (Made\_in = japan)$. The new body $Made\_in = japan$ covers the following five positive instances and one negative instance.

ID	class	$Made\_in$	$Size$	$Price$
1	positive	$japan$	$small$	$cheap$
2	positive	$japan$	$small$	$middle$
3	positive	$japan$	$small$	$expensive$
4	positive	$japan$	$middle$	$cheap$
5	positive	$japan$	$middle$	$expensive$
6	negative	$japan$	big	$middle$

Because the value of $Made\_in$ is fixed to $japan$, IGR generates four conditions on other two attributes to reject the negative instance and evaluates them as follows.

condition	estimation
$Size \in \{small, middle\}$	0.65
$Price \in \{cheap, expensive\}$	0.32
$Size = big$	-0.32
$Price = middle$	-0.65

All of the target instances in $Targets$ satisfy the best condition $Size \in \{small, middle\}$ that covers only positive instances. IGR is called with $Body = (Made\_in = japan \wedge Size \in \{small, middle\})$ and discovers the following rule.

$$Made\_in = japan \wedge Size \in \{small, middle\} \Rightarrow class = positive$$

As a result, IGR with $Body = (Made\_in = japan)$ returns a rule set with only this rule.

After the recursive call with $Body = (Made\_in = japan)$, IGR updates weights of instances with the given parameter α. Let us assume $\alpha = 0.5$. Then, the weights become 0.5 for instances 1$\sim$5 and 1.0 for others.

Next, IGR is called with $Targets = \{7\}$ and $Body = (Size = small)$. Because the instances 1,2,3,7 and 8 satisfy $Body$, the weighted class distribution for each attribute value becomes

	positive	negative
$Made\_in = japan$	1.5	0.0
$Made\_in = europe$	1.0	1.0
$Price = cheap$	0.5	0.0
$Price = middle$	1.5	0.0
$Price = expensive$	0.5	1.0

and IGR generates the following four conditions

condition	estimation
$Price \in \{cheap, middle\}$	0.46
$Maid\_in = japan$	0.16
$Maid\_in = europe$	-0.16
$Price = expensive$	-0.46

Because there is only one target instance 7 and this instance satisfies the first condition, $Price \in \{cheap, middle\}$, the recursive call occurs only for this condition and returns the following rule.

$$Size = small \wedge Price \in \{cheap, middle\} \Rightarrow class = positive.$$

At the end, IGR is called for the target instance 12 with $Body = (Price = middle)$ and returns

$$Made\_in = europe \wedge Price = middle \Rightarrow class = positive.$$

As a result, IGR learns the following three rules for class *positive* from the training instances shown in Table 2.1.

$$R_1 : Made\_in = japan \wedge Size \in \{small, middle\} \Rightarrow class = positive,$$

$$R_2 : Size = small \wedge Price \in \{cheap, middle\} \Rightarrow class = positive,$$

$$R_3 : Made\_in = europe \wedge Price = middle \Rightarrow class = positive.$$

Table 2.2: Characteristics of databases.

	classes	attributes	instances		classes	attributes	instances
breast	2	9	638	promoters	2	57	106
crx	2	15	653	satellite	7	36	6,435
dna	3	60	3,186	segment	7	19	2,310
glass	7	10	214	soy-large	19	35	307
hayes-roth	4	4	160	tic-tac-toe	2	9	958
iris	3	4	351	vehicle	4	18	846
krkp	2	36	3,196	voting	2	16	435
monks-1	2	6	432	waveform	3	21	5,000
pima	2	8	768	wine	3	13	178

After inducing rules, IGR assigns weights of rules. IGR requires a set of instances covered by a rule r, $I(r)$ and a set of rules that cover an instance i, $S(i)$ to calculate the weight of r. The first rule R_1 covers five instances, 1$\sim$5 and the instance 1 and 2 are also covered by R_2 but the remaining three instances are covered by only R_1. Then, the weight for R_1 is calculated as follows.

$$\begin{aligned} weight(R_1) &= \sum_{i \in \{1,2,3,4,5\}} \frac{1}{|S(i)|} \\ &= \frac{1}{2} + \frac{1}{2} + \frac{1}{1} + \frac{1}{1} + \frac{1}{1} \\ &= 4.0. \end{aligned}$$

Similarly, the weights for R_2 and R_3 are

$$\begin{aligned} weight(R_2) &= 2.0, \\ weight(R_3) &= 1.5. \end{aligned}$$

2.3 Experiments

This section compares IGR with AQ, C4.5 and two bagging learners with C4.5 by using eighteen databases in UCI machine learning repository[6]. AQ has many variants

to induce a rule for one seed instance. To show how IGR improves classification performance by learning more rules than AQ, this section uses AQ with the same conditions for specialization and the same condition selection criterion used in IGR. The bagging learners use C4.5 as their weak learner and learn five and twenty decision trees respectively.

Table 2.2 shows the domain characteristics of the databases used in the experiments. Because IGR can't deal with numeric attribute directly, ranges of numeric attributes were discretized into five intervals with same population for small databases with less than 500 instances and were discretized into ten intervals for larger databases with more than 500 instances. AQ also learned from discretized instances because it used the same conditions for specialization with IGR. Other learning algorithms, C4.5 and bagging with it, can deal with numeric attributes directly and learned rules from original, non-discretized instances. All experiments in this section rejected instances with missing attribute value and used only instances in which values for all attributes are given.

Before comparing IGR with other learning algorithms, I first investigate how weighting of instances with α affects discovered rule sets and how weights of rules in majority voting affects classification accuracy. Table 2.3 reports the effects of α on classification accuracy. The results in this table are the average of ten runs of five fold cross validation. In most databases, classification accuracy slightly decreases as α decreases but the differences were quite small when $\alpha \geq 0.05$. Then, from the viewpoint of classification accuracy, there is no reason to assign weights to the training instances.

However, weighting instances with α is important to learn smaller hypotheses with shorter learning time. Table 2.4 shows how α affects the hypothesis size, i.e. the number of rules, and learning time. For example, IGR with $\alpha = 0.1$ extracted about

Table 2.3: Dependency of classification accuracy on α. All results are the average of ten runs of 5-fold cross validation and this table also reports 95% confidence intervals of the average accuracy.

	α=1.00	α=0.20	α=0.10	α=0.05	α=0.00
breast	95.9±0.2	95.8±0.1	95.7±0.2	95.7±0.2	95.2±0.3
crx	86.3±0.4	85.5±0.5	85.3±0.3	85.6±0.4	83.8±0.6
dna	96.0±0.1	96.1±0.1	95.9±0.1	95.9±0.1	94.9±0.1
glass	69.2±1.3	68.1±1.3	67.9±0.9	68.0±1.1	68.2±1.1
hayes-roth	82.8±0.9	82.6±0.9	82.6±0.9	82.7±1.0	82.3±1.1
iris	90.7±0.9	90.6±0.8	90.7±0.8	90.8±0.9	91.1±0.9
krkp	99.0±0.1	99.2±0.1	99.2±0.1	99.3±0.1	99.4±0.1
monks-1	97.6±1.1	96.6±0.8	96.5±1.1	96.5±1.1	96.8±0.9
pima	74.4±0.6	74.9±0.4	74.6±0.5	74.1±0.7	72.4±0.8
promoters	87.5±1.4	86.6±1.3	84.9±1.3	83.3±1.7	80.1±2.1
satellite	85.6±0.1	86.7±0.1	86.1±0.2	85.7±0.3	83.0±0.2
segment	95.8±0.2	96.1±0.1	95.9±0.1	95.8±0.1	94.3±0.2
soy-large	85.8±0.8	85.8±0.9	85.7±0.7	85.3±0.7	84.7±0.8
tic-tac-toe	96.3±0.5	97.8±0.2	97.6±0.2	97.4±0.3	97.4±0.3
vehicle	69.8±0.6	69.4±0.8	68.5±0.9	68.3±1.0	64.5±1.2
voting	94.6±0.3	94.7±0.3	94.8±0.3	94.7±0.3	94.9±0.2
waveform	82.6±0.2	82.0±0.2	81.0±0.2	80.1±0.2	74.9±0.3
wine	93.3±0.6	93.0±0.6	92,7±0.4	92.4±0.5	91.0±0.6

30% less rules in about 30% shorter learning time than IGR without weighting, i.e. $\alpha = 1$. This difference is not so large on average but the effects of the weights became significant in the databases in which IGR discovered many rules such as satellite and waveform. In these databases, IGR with weighting instances induced less than a half rules and was about three times faster than IGR without weighting. Then, weighting with small α is useful from the viewpoints of simplicity of learned hypotheses and the complexity for learning. In the following experiments in this section, α is fixed at 0.1.

Table 2.5 shows the effect of the weights of rules in majority voting defined in equation (2.3). This table compares three kinds of weights for rules in majority voting, constant weight for every rule, using support of rules as the weights of rules and the

Table 2.4: Dependency of the number of rules and learning time on α. All results are the average of ten runs of 5-fold cross validation. Small α decreased the number of induced rules and improved running time.

	# of rules					learning time(cpu sec)				
	1.00	0.20	0.10	0.05	0.00	1.00	0.20	0.10	0.05	0.00
breast	36	29	28	26	25	0.22	0.17	0.16	0.15	0.13
crx	109	77	71	66	62	0.59	0.39	0.34	0.32	0.27
dna	232	183	163	154	134	9.86	8.25	7.49	6.96	5.02
glass	72	59	56	55	54	0.16	0.12	0.11	0.11	0.10
hayes-roth	13	13	13	13	13	0.04	0.03	0.03	0.03	0.02
iris	17	16	16	16	15	0.02	0.02	0.02	0.02	0.02
krkp	96	66	61	59	58	3.31	2.33	2.18	2.04	1.59
monks-1	72	56	56	56	49	0.15	0.10	0.10	0.10	0.90
pima	243	145	131	126	122	0.74	0.42	0.37	0.34	0.29
promoters	14	12	12	12	11	0.09	0.08	0.08	0.08	0.09
satellite	1,683	846	735	686	630	84.58	28.72	23.21	20.60	16.53
segment	166	116	108	104	100	2.22	1.59	1.45	1.34	1.09
soy-large	47	42	41	41	41	0.38	0.34	0.33	0.33	0.32
tic-tac-toe	182	103	95	92	90	0.56	0.31	0.27	0.26	0.24
vehicle	253	151	143	139	135	1.65	0.85	0.77	0.74	0.66
voting	32	27	25	25	25	0.18	0.15	0.14	0.17	0.11
waveform	1,496	850	688	621	574	30.78	15.03	10.28	7.74	5.84
wine	16	15	14	14	14	0.04	0.04	0.04	0.03	0.03

weighted support defined in equation (2.3). In most databases, weighting rules with their support degraded classification ability of induced rules compared with other two weights. Weighted support was comparable to constant weight in many databases but was superior than constant weight in monks-1 and tic-tac-toe.

Table 2.6 and 2.7 compare IGR with other learning algorithms. Table 2.6 shows average classification accuracy and 95% confidence intervals over ten runs of five-fold cross validation. Table 2.7 shows the average number of discovered rules. For C4.5 and bagging learners, this table reports the number of leaf nodes of decision trees as the number of rules because a path from root to each leaf represents one if-then rule.

Table 2.5: Classification accuracy from three kinds of weights of rules, constant weight, support and weighted support that IGR applies.

	constant	support	weighted sup.
breast	96.1±0.2	95.7±0.3	95.7±0.2
crx	85.0±0.5	82.5±0.9	85.5±0.3
dna	95.8±0.1	95.1±0.2	95.9±0.1
glass	67.5±1.5	65.5±1.6	67.9±0.9
hayes-roth	83.4±1.2	84.1±0.9	82.7±0.9
iris	90.7±0.8	89.7±0.7	90.7±0.8
krkp	99.3±0.1	99.2±0.1	99.2±0.1
monks-1	92.5±1.3	89.2±1.8	96.4±1.1
pima	74.2±0.5	72.0±0.7	74.6±0.5
promoters	85.2±1.2	85.3±1.6	84.9±1.3
satellite	86.2±0.3	83.2±0.1	86.2±0.2
segment	95.9±0.2	95.0±0.1	95.9±0.1
soy-large	85.0±0.9	85.3±1.0	85.7±0.7
tic-tac-toe	95.9±0.3	94.1±0.5	97.6±0.2
vehicle	68.0±0.9	64.7±0.9	68.5±0.9
voting	94.6±0.4	93.7±0.5	94.8±0.3
waveform	79.9±0.2	78.1±0.3	80.1±0.2
wine	93.0±0.6	92.6±0.6	92.7±0.4

By comparing with AQ, IGR achieved much higher accuracy in most databases. IGR lost only one database, krkp, but IGR also achieved more than 99% accuracy in this database. The poor results of AQ in small databases with numeric attributes, such as iris and wine, were mainly caused by the discretization of the attributes. Even if each discretized value involves sufficiently many training instances before learning, some discretized values become empty, or involve only a few instances after specialization going on. This causes too specific rules and degrades the classification accuracy of learned rules.

IGR also outperformed C4.5 with respect to classification accuracy in ten databases (the difference of average accuracy was larger than 95% confidence interval) and the

Table 2.6: Classification accuracy and its 95% confidence interval. Bagging(5) and bagging(20) use C4.5 as their weak learner and learn five and twenty decision trees respectively.

	AQ	C4.5	bagging(5)	bagging(20)	IGR($\alpha = 0.1$)
breast	82.9±3.6	93.3±0.5	94.4±0.4	94.8±0.5	95.7±0.2
crx	84.8±0.5	83.9±0.8	85.2±0.4	86.4±0.3	85.3±0.3
dna	83.5±1.7	93.8±0.2	94.1±0.2	94.6±0.2	95.9±0.1
glass	57.7±2.1	66.7±1.6	68.1±1.7	71.1±0.7	67.9±0.9
hayes-roth	70.1±1.9	74.0±2.0	76.7±1.4	76.7±1.6	82.6±0.9
iris	68.0±2.9	94.0±0.6	94.9±0.6	94.5±0.5	90.7±0.8
krkp	99.6±0.1	99.4±0.1	99.4±0.0	99.5±0.0	99.2±0.1
monks-1	85.7±4.6	91.2±1.9	93.5±1.4	96.4±1.3	96.5±1.1
pima	71.1±0.6	72.3±1.0	73.2±0.5	74.7±0.6	74.6±0.5
promoters	51.7±4.2	76.9±1.1	81.4±1.6	83.9±1.5	84.9±1.3
satellite	73.5±0.3	85.1±0.2	86.9±0.2	88.6±0.1	86.2±0.2
segment	70.9±2.4	96.0±0.2	96.0±0.2	96.3±0.1	95.9±0.1
soy-large	69.6±1.4	88.7±0.9	88.5±0.6	90.0±0.7	85.7±0.7
tic-tac-toe	91.0±1.2	84.8±0.5	89.7±0.8	92.2±0.6	97.6±0.5
vehicle	56.3±1.4	70.9±0.5	72.8±0.8	74.2±0.4	68.5±0.9
voting	92.1±1.7	94.9±0.3	95.4±0.3	95.4±0.2	94.8±0.3
waveform	67.7±0.6	72.9±0.3	77.7±0.3	80.2±0.3	81.0±0.2
wine	60.3±4.8	92.1±1.3	94.2±0.8	95.6±0.9	92.7±0.4

difference between them were about 10% in hayes-roth, tic-tac-toe and waveform. IGR lost only six databases but the differences were at most 3.7%. This result shows IGR can induce more accurate rule sets than C4.5 in many databases.

Bagging improved C4.5 in most databases. This section tried bagging with five and twenty decision trees. In most databases, bagging with more trees achieved higher classification accuracy. By comparing IGR with bagging, IGR won in some databases such as hayes-roth and tic-tac-toe, but lost glass, iris, satellite and vehicle databases. The differences were not so large in other databases. From the view point of classification accuracy, IGR and bagging were also better than AQ and C4.5. However, bagging with twenty trees induced about ten times more rules than IGR. In other words, IGR

Table 2.7: Number of discovered rules.

	AQ	C4.5	bagging(5)	bagging(20)	IGR(α−0.1)
breast	7	25	135	532	28
crx	20	38	157	623	71
dna	31	104	613	2,449	163
glass	17	37	137	547	56
hayes-roth	8	14	89	358	13
iris	7	4	21	85	16
krkp	25	29	151	610	61
monks-1	8	30	181	712	56
pima	32	94	365	1,449	131
promoters	4	12	52	208	12
satellite	97	429	1,776	7,122	735
segment	23	59	244	978	108
soy-large	20	40	181	726	41
tic-tac-toe	19	85	413	1,653	95
vehicle	27	122	439	1,757	143
voting	11	6	36	148	25
waveform	76	638	1,998	7,997	688
wine	4	6	30	120	14

achieved similar classification accuracy with bagging with twenty trees with only 10% of rules.

Table 2.8 shows average running time on sun workstation with 300MHz Ultra-SPARCII. Even if IGR and bagging with twenty decision trees were comparable with respect to classification accuracy, IGR was about 5∼90 times faster than bagging and was comparable to C4.5.

2.4 Chapter Summary

This chapter discussed discover of classification rules for predicting classes of new instances correctly. In discovery for prediction, classification performance is the primary criterion of discovered rules. In addition, the simplicity of learned rules is also im-

Table 2.8: Average learning time (cpu seconds) of the three algorithms, C4.5, bagging with 20 runs of C4.5 and IGR.

	C4.5	bagging(20)	IGR(α=0.1)
breast	0.07	1.18	0.16
crx	0.67	8.72	0.34
dna	2.69	48.16	7.45
glass	0.40	5.48	0.11
hayes-roth	0.02	0.27	0.03
iris	0.02	0.33	0.02
krkp	1.31	26.03	2.18
monks-1	0.04	0.81	0.10
pima	1.55	18.91	0.37
promoters	0.09	1.39	0.08
satellite	57.79	875.45	23.21
segment	5.46	85.66	1.45
soy-large	0.30	5.36	0.33
tic-tac-toe	0.16	2.97	0.27
vehicle	2.51	36.76	0.77
voting	0.07	1.13	0.14
waveform	82.64	940.06	10.28
wine	0.13	2.04	0.04

portant because the simple rules are easy to understand. This chapter presented a new discovery algorithm IGR that achieves high classification performance without increasing the number of rules. IGR learns the pseudo-optimum rule for each training instance that covers as many positive instances as possible and rejects all negative instances.

Empirical results show that IGR can learn much more accurate rule sets than AQ and C4.5 in many databases. IGR is comparable to bagging with C4.5 with respect to classification accuracy but the size of rule sets from IGR is quite smaller than the sets from bagging. In addition, IGR is much faster than bagging learners.

Even if IGR achieved high accuracy, there are some remaining problems to be improved. The most important problem is pruning[16, 28] learned rules. The current IGR doesn't apply pruning and generates only 100% accurate rules. The lack of pruning tends to lead over specialized rules especially in noisy databases and degrades the ability of classification. In addition, it increases the number of rules and degrades readability of learned rules.

Chapter 3

Discovery of Interesting Rules in Nominal Domains

3.1 Classification Rules in Nominal Domains

This chapter investigates rule discovery for analyzing characteristics of given databases. This chapter discusses what kind of rules are useful and presents the interestingness of rules as the criterion of classification rules. This chapter also presents a new discovery algorithm based on the interestingness. This chapter assumes all attributes are nominal and deals with classification rules with grouping attribute values[22, 44] like

$$x_1 \in D_1 \wedge \cdots \wedge x_L \in D_L \Rightarrow class = c, \tag{3.1}$$

where x_i is an attribute and D_i is a subset of possible values for x_i. This rule means if a value of a certain instance for attribute x_i is a member of D_i for every $1 \leq i \leq L$, then the instance belongs to class c. For simplicity, I will use $x_i = v$ instead of $x_i \in D_i$ if D_i includes only one value v. Many learning/discovery algorithms[51, 62] deal with only a rule without grouping attribute values that permits exactly one value for each attribute. This thesis calls such rules primitive rules. Grouping values increases the number of possible rules and makes rule discovery difficult. Instead, it increases readability of discovered rules because one rule with grouping can represent plural primitive rules.

3.2 Interestingness of Classification Rules

3.2.1 Limitation of Rule Discovery based on Support and Accuracy

This subsection shows why it is not sufficient to restrict support and accuracy to extract interesting rules. Support of a classification rule is the probability of instances that belong to the class in its head and satisfy all conditions in its body. The support of the rule 3.1 is

$$P(class = c \wedge x_1 \in D_1 \wedge \cdots \wedge x_L \in D_L). \tag{3.2}$$

Accuracy of a rule is the conditional probability of the class of its head on the condition that its body is satisfied,

$$P(class = c | x_1 \in D_1 \wedge \cdots \wedge x_L \in D_L). \tag{3.3}$$

If support of a rule is quite low, then the rule represents very rare case and it can be applied with very small probability. If accuracy of a rule is low, then we can not believe the results implied with the rule. Both of support and accuracy are fundamental characteristics of classification rules and it is natural to require large support and high accuracy for rules to be discovered.

Many discovery systems extract rules whose support and accuracy are higher than the given lower bounds[1]. However, the fixed lower bounds for support and accuracy are not sufficient to filter out useless rules. Following is an example of a rule discovered from mushroom database in UCI machine learning repository[6]. This database includes about 8,000 mushrooms as instances and each instance is classified into *edible* or *poisonous*.

$$R1 : odor \in \{almond, none\} \wedge cap\_color \in \{brown, red\} \Rightarrow edible,$$

where

$$support(R1) = 23\%,$$

$$accuracy(R1) = 99\%.$$

Support and accuracy of this rule are 23% and 99% respectively and this rule looks like a quite good rule because of its large support and high accuracy. However, this rule is useless in usual because the following rule can be also discovered from the database.

$$R2 : odor \in \{almond, none\} \Rightarrow edible,$$

$$support(R2) = 47\%, accuracy(R2) = 97\%.$$

This rule is a generalization of $R1$ and can explain class labels of all edible mushrooms covered by $R1$ with similar confidence. The specialization of $R2$ to $R1$ by the condition $cap\_color \in \{brown, red\}$ rejects a half of positive instances covered by $R2$ and gains only 2% for accuracy. Unless this slight gain of accuracy is quite important for the purpose of rule discovery, $R1$ is a trivial conclusion of $R2$ and it is sufficient to discover only $R2$.

This example is not a rare case. If there exist a rule with large support and high accuracy, specializing it with irrelevant conditions with no relationship to classes causes many useless rules with large support and high accuracy. To reject such rules, it is not sufficient to constrain rules only with their support and accuracy, and different criteria are required to extract interesting rules.

Following is also an example of a rule from mushroom database.

$$R3 : stalk\_shape = enlarging \wedge habitat = woods \Rightarrow poisonous,$$

$$support(R3) = 9\%, accuracy(R3) = 93\%.$$

Its generalizations conclude *poisonous* with the following probabilities.

$$Accuracy(stalk\_shape = enlarging \Rightarrow poisonous) = 54\%,$$

$$Accuracy(habitat = woods \Rightarrow poisonous) = 41\%.$$

Because about 48% of mushrooms in the database are poisonous, these probabilities mean that every condition in the body of $R3$ has only very weak relation to whether *poisonous* or *edible*, but the rule $R3$ shows the combination of the conditions leads poisonous with high accuracy. In other words, both of the conditions are required to conclude poisonous. Even if both of support and accuracy of $R3$ are lower than those of $R1$, this rule is usually more preferable than $R1$ because the accuracy of its generalizations are much lower than the accuracy of $R3$ and $R3$ gives us new information for poisonous mushrooms even if we already know the general rules.

The above example shows that interestingness of a rule strongly depends on more general and simpler rules. If a rule concludes the same class with similar accuracy with its generalizations, then the rule gives no new information and is useless. Oppositely, if accuracy of the rule is much higher than all of its generalizations and all conditions in the rule are required to imply the conclusion, then it is worth to show the rule to the users of discovery systems.

3.2.2 Interestingness of Primitive Rules

This subsection discusses interestingness of primitive rules before rules with grouping attribute values and proposes a criterion of primitive rules that evaluates whether every condition in a rule to be evaluated is really required to lead the conclusion of the rule. The next subsection will modify it to evaluate rules with grouping values.

Let's consider the rule in (3.1) with L conditions. This subsection assumes primitive rules and each D_i involves exactly one value for attribute x_i. For simplicity, let us use B to represent the body of the rule and B_i to represent a conjunction of $L-1$ conditions

in B excluding the ith condition, i.e.

$$B_i = x_1 \in D_1 \wedge \cdots \wedge x_{i-1} \in D_{i-1} \wedge x_{i+1} \in D_{i+1} \wedge \cdots \wedge \wedge x_L \in D_L. \tag{3.4}$$

As described above, I require interesting rules that all conditions in their bodies are required to achieve high accuracy. This leads the following constraint.

$$P(c|B) > P(c|B_i), \ \ 1 \leq \forall i \leq L. \tag{3.5}$$

This constraint means every condition $x_i \in D_i$ in B contributes to improve accuracy of rule (3.1). The more the difference of accuracy between the rule (3.1) and its generalization($B_i \Rightarrow c$) is, the more the condition is important to conclude class c.

The constraint (3.5) is not sufficient to reject useless rules that are trivial conclusions of more general rules. When a condition $x_i \in D_i$ positively relates to class c, it is not interesting even if the specialization by the condition increases the probability of c and the constraint (3.5) is satisfied. Assuming independence of $x_i \in D_i$ and B_i, accuracy of the rule (3.1) can be calculated as follows.

$$\begin{aligned} P(c|B) &= P(c|x_i \in D_i \wedge B_i) \\ &= \frac{P(x_i \in D_i \wedge B_i|c)P(c)}{P(x_i \in D_i \wedge B_i)} \\ &= \frac{P(x_i \in D_i|c)P(B_i|c)P(c)}{P(x_i \in D_i)P(B_i)} \\ &= \frac{P(c|x_i \in D_i)P(c|B_i)}{P(c)}. \end{aligned}$$

If the real accuracy is comparable to this expectation, the rule to be evaluated represents only a regularity that can be easily expected from two more general rules, $x_i \in D_i \Rightarrow c$ and $B_i \Rightarrow c$. Then, it is natural to require the following constraint in addition to (3.5).

$$P(c|B) > \frac{P(c|x_i \in D_i)P(c|B_i)}{P(c)}, \ \ 1 \leq \forall i \leq L. \tag{3.6}$$

Again, large difference between the left hand side and the right hand side means large contribution of the condition $x_i \in D_i$ to the rule.

Both of the constraints (3.5) and (3.6) give lower bounds of accuracy and we define interestingness of a primitive rule as the margin of its accuracy to satisfy these constraints as follows.

$$I(B \Rightarrow c) = \frac{P(c|B) - \max\limits_{1 \leq i \leq L} \left(P(c|B_i), \min\left(1, \frac{P(c|x_i \in D_i)P(c|B_i)}{P(c)}\right)\right)}{1 - P(c)}. \tag{3.7}$$

The denominator, $1 - P(c)$, is a normalization factor for class distributions and the interestingness becomes 1 for the rule that concludes a certain class with 100% accuracy even if each its generalization, $x_i \in D_i$ and B_i, is independent of the conclusion. The constraint on the minimum interestingness restricts accuracy of rules with dynamic, not fixed, lower bounds depending on accuracy of their generalizations.

For a rule with only one condition, its generalization is the rule with no condition and its interestingness becomes

$$I(x_1 \in D_1 \Rightarrow c) = \frac{P(c|x_1 \in D_1) - P(c)}{1 - P(c)}, \tag{3.8}$$

and the restriction on the interestingness coincides with the restriction of accuracy with a fixed threshold.

Let me show how this interestingness works for the examples $R1$ and $R3$ from mushroom database. Because 48% of mushrooms in the database are poisonous and the relationships between poisonous and the conditions in $R1$ are

$$P(poisonous|odor \in \{almond, none\}) = 97\%,$$

$$P(poisonous|cap\_color \in \{brown, red\}) = 50\%.$$

Then the interestingness of $R1$ is

$$\begin{aligned} I(R1) &= \frac{0.99 - \max(0.97, 0.50, \frac{0.97 \times 0.50}{0.48})}{1 - 0.48} \\ &= -0.03. \end{aligned}$$

This means the interestingness I judges $R1$ is not interesting at all. In opposite, the interestingness of $R3$ becomes

$$\begin{aligned} I(R1) &= \frac{0.93 - \max(0.54, 0.41, \frac{0.54 \times 0.41}{0.48})}{1 - 0.48} \\ &= 0.75, \end{aligned}$$

and $R3$ is interesting with respect to I. These results for $R1$ and $R3$ coincide with the discussion in the previous subsection.

3.2.3 Interestingness of Rules with Grouping Attribute Values

The interestingness I defined in (3.7) works well for primitive rules as shown in the previous subsection but it sometimes gives high scores to inappropriate rules with grouping attribute values. Following is an example from mushroom database again.

$$R4 : cap\_color \in \{brown, red\} \wedge stalk\_root \in \{bulbous, rooted\} \Rightarrow edible,$$

where

$$Support(R4) = 16\%, Accuracy(R4) = 100\%.$$

This rule looks like interesting because accuracy of its generalizations are

$$Accuracy(cap\_color \in \{brown, red\} \Rightarrow edible) = 50\%,$$

$$Accuracy(stalk\_root \in \{bulbous, rooted\} \Rightarrow edible) = 53\%,$$

and every condition in $R4$ has very weak relation to the conclusion of the rule but only the edible instances satisfy the conjunction of the two conditions. The interestingness defined in (3.7) gives high score, 0.98 to $R4$ and judges $R4$ is interesting. However, by exploring relationship between *edible* and each attribute value allowed in the rule, we

can find the following rule.

$$stalk\_root = rooted \Rightarrow edible,$$

$$Accuracy = 100\%.$$

This rule means that the condition on *cap_color* in $R4$ is redundant for mushrooms with $stalk\_root = rooted$ and $R4$ is not interesinting at all for classifying the mushrooms. Instead, the following rule that prohibits $stalk\_root = rooted$ is more preferable than $R4$ because each condition in its body is required to classify *all* mushrooms covered by its body with high accuracy.

$$R5 : cap\_color \in \{brown, red\} \wedge stalk\_root = bulbous \Rightarrow edible.$$

The problem of the interestingness defined in (3.7) is that it evaluates a rule based on only each condition in its body and it does not concern each attribute value allowed in the conditions at all. To resolve this problem, we introduce the interestingness of an attribute value to evaluate how the condition on the attribute contributes to classify the instances with the value accurately. Formally, the interestingness of a value v for an attribute x_i allowed in the rule $x_1 \in D_1 \wedge \cdots \wedge x_L \in D_L \Rightarrow c$ is

$$\begin{aligned} &I_{value}(x_i = v, x_1 \in D_1 \wedge \cdots \wedge x_L \in D_L \Rightarrow c) \\ &= \max_{v_k \in D_k (k \neq i)} I(x_1 = v_1 \wedge \cdots \wedge x_i = v \wedge \cdots \wedge x_L = v_L \Rightarrow c). \end{aligned} \tag{3.9}$$

For example, the interestingness of the attribute values in $R4$ is calculated from the interestingness of the following four primitive rules.

$$cap\_color = brown \wedge stalk\_root = bulbous \Rightarrow edible\ (I = 0.93),$$

$$cap\_color = brown \wedge stalk\_root = rooted \Rightarrow edible\ (I = 0.00),$$

$$cap\_color = red \wedge stalk\_root = bulbous \Rightarrow edible\ (I = 1.02),$$

$$cap\_color = red \wedge stalk\_root = rooted \Rightarrow edible\ (I = 0.00).$$

Stalk_root=bulbous appears in the first and the third rule and its interestingness is 1.02. *Stalk_root=rooted* appears in the second and the last rule and its interestingness becomes 0.

Each instance covered by the original rule with grouping values is covered by exactly one primitive rule and high score of $I_{value}(x_i = v, R)$ means that there is at least one instance with $x_i = v$ for which R works better than expected from more general rules. Oppositely, $I_{value}(x_i = v, R)$ becomes low when more general rules can classify all instances with $x_i = v$ with comparable or higher accuracy than R.

Using the interestingness of attribute value defined in (3.9), the interestingness of a rule, I, is modified to evaluate rules with grouping values as follows.

$$IG(R) = \min\left(I(R), \min_{1 \leq k \leq L, v \in D_k} I_{value}(x_k = v, R)\right), \tag{3.10}$$

where R is the following rule with grouping attribute values.

$$x_1 \in D_1 \wedge \cdots \wedge x_L \in D_L \Rightarrow class = c. \tag{3.11}$$

This new criterion of interestingness coincides with the old one, I, for primitive rules but gives lower scores than I to the rules with grouping values that permit attribute values with low interestingness. In other words, the new interestingness IG gives a high score to only a rule in which there is at least one instance with each allowed value for which the rule works much better than its generalizations.

3.3 Discovery Algorithm: DIG

This section proposes a new rule discovery algorithm DIG(Discover Interesting rules with Grouping attribute values) to extract rules with high interestingness with respect to IG defined in (3.10). For simplicity, I assume that the conclusion and the set of L attributes in bodies of rules are given. Therefore, extracting rules is a task to

select allowed values for each given attribute. It is possible to enumerate all rules with L attributes by applying the algorithm to all combinations of class labels and L attributes. This section also assumes all attributes are nominal and values of all instances for all attributes are known. The inputs of the algorithm are a set of training instances, a class label as a conclusion, a set of L attributes, a lower bound of support, LB_{sup}, a lower bound of accuracy, LB_{acc} and a lower bound of the interestingness with respect to IG, LB_{int}. The discovery algorithm outputs a set of rules with the given L attributes in their bodies whose support, accuracy and the interestingness are higher than or equal to LB_{sup}, LB_{acc} and LB_{int} respectively.

One approach to discover interesting rules is an apriori[1] like algorithm that first enumerates all rules with large support and filters out rules whose accuracy or the interestingness is lower than the given threshold. This approach is practical to extract primitive rules but may be too inefficient for rules with grouping values. This is because grouping attribute values drastically increases the number of possible rules and the first step of this approach, enumeration of rules with large support, requires huge time even when L is small. For example, when extracting rules with a given set of three attributes with ten values, there is only 10^3 primitive rules for each class but the number of rules with grouping is $(2^{10} - 2)^3 \simeq 10^9$. In usual, the constraint of minimum support can delete only rules with a few allowed values for each attribute and most rules satisfy the constraint on their support. However, we don't need to enumerate all rules with large support to extract rules with high interestingness. Because of the definition of IG in (3.10), the constraint of minimum interestingness

$$IG\left(x_1 \in D_1 \wedge \cdots \wedge x_L \in D_L \Rightarrow class = c\right) \geq LB_{int}$$

requires

$$1 \leq \forall k \leq L, \forall v \in D_k,$$

procedure DIG($Training, target, (x_1, \ldots, x_L), LB_{sup}, LB_{acc}, LB_{int}$)

inputs:

$Training$	: set of training instances.
$target$	: class label (head) of the rules to be discovered.
$(x_1, \ldots, x_L)$	: set of attributes in bodies of rules.
$LB_{sup}, LB_{acc}, LB_{int}$	: lower bounds of support, accuracy and interestingness.

output:

$Rules$: set of discovered rules.

1. Count class distribution
   $P(target|x_1 = v_1, \ldots, x_L = v_L)$
   for each combination of values $(v_1, \ldots, v_L)$ for the given attributes $(x_1, \ldots, x_L)$.
2. $S := \phi$.
3. **Foreach** combination of values $(v_1, \ldots, v_L)$
4. **If** $I\,(x_1 = v_1 \wedge \cdots \wedge x_L = v_L \Rightarrow target) \geq LB_{int}$ **then**
5. $S := S \cup \{x_1 = v_1 \wedge \cdots \wedge x_L = v_L \Rightarrow target\}$.
6. $Rules := \phi$.
7. **Foreach** $S' \subseteq S$
8. $D_k := \{v | r \in S' \wedge r$ permits $x_k = v\},\ k = 1, \ldots, L$.
9. $R := "x_1 \in D_1 \wedge \cdots \wedge x_L \in D_L \Rightarrow target"$.
10. **If** $support(R) \geq LB_{sup}$ and $accuracy(R) \geq LB_{acc}$ and $IG(R) \geq LB_{int}$
11. $Rules := Rules \cup \{R\}$.
12. **Return** $Rules$.

Figure 3.1: Discovery algorithm DIG.

$$\exists v_1 \in D_1, \ldots, \exists v_{k-1} \in D_{k-1}, \exists v_{k+1} \in D_{k+1}, \ldots, \exists v_L \in D_L, \text{s.t.}$$

$$I(x_1 = v_1 \wedge \cdots \wedge x_{k-1} = v_{k-1} \wedge x_k = v \wedge x_{k+1} = v_{k+1} \wedge \cdots \wedge x_L = v_L \Rightarrow c) \geq LB_{int}.$$

This property leads the following efficient algorithm that first enumerates all *primitive* rules whose interestingness is higher than or equals to LB_{int} and generates rules with grouping by combining the primitive rules with high interestingness. Because the number of primitive rules is much smaller than the number of rules with grouping, this approach can effectively prune rules with grouping that don't satisfy the constraint on the interestingness.

Figure 3.1 shows a pseudo-code of the discovery algorithm DIG. DIG first counts

class distribution $P(c|x_1 = v_1, \ldots, x_L = v_L)$ for every combination of values for the given attributes (step 1). Based on these distributions, DIG evaluates the interestingness of all primitive rules and selects a set S of primitive rules whose interestingness is higher than or equals to the given lower bound LB_{int} (step 2∼5). For each subset S' of S, DIG generates a rule R with grouping values that allows attribute values appearing in at least one primitive rule in S'. DIG checks whether R satisfies the constraints on its support, accuracy and the interestingness with respect to IG, and adds it to a set of discovered rules, $Rules$, if R satisfies all of them (step 7∼11). DIG can enumerate all rules with grouping values with large interestingness efficiently because a rule with $IG \geq LB_{int}$ allows only attribute values appeared in at least one primitive rule with $I \geq LB_{int}$.

3.4 Time Complexity

This subsection discusses the time complexity of DIG and explores when DIG can extract rules efficiently and when DIG requires huge running time. Let K, M, D and N be the number of classes, the number of attributes, the average number of possible values for each attribute and the number of training instances respectively.

Roughly speaking, DIG consists of three main processes, counting class distribution for every combination of values of L attributes, enumerating primitive rules with high interestingness, and generating rules with grouping from the primitive rules. In the first process, DIG checks values of L attributes for all training instances and the complexity of of this step is $O(N \cdot L)$. The second process requires $O(D^L \cdot L)$ because there are D^L primitive rules when L attributes and the target class are given, and evaluation of interestingness for one primitive rule requires to evaluate accuracy of L general rules of it. The complexity of the last process is $O(2^{|S|} \cdot D^L)$ where S is a set of primitive rules

with high interestingness. In this process, DIG uses the table of class distributions generated in the first step to evaluate rules with grouping. For evaluating a rule with grouping, DIG sums up the distributions for all combinations of values allowed in the rule to calculate support and accuracy of the rule and the complexity to evaluate one rule is at most D^L because the number of allowed value combinations is at most D^L. Because DIG evaluates one rule with grouping for each subset of S and the number of subsets is $2^{|S|}$, the complexity of the last process becomes $O(2^{|S|} \cdot D^L)$.

When the lower bound of IG is quite low and S involves many primitive rules, the most computationally heavy process is the last process and DIG becomes impractical. However, the number of primitive rules with high interestingness rapidly decreases as LB_{int} increases, and the first process, counting class distributions, becomes more heavy than the last process. In this case, the total complexity of extracting rules with L attributes is $O\left(K \cdot \left(\begin{array}{c} M \\ L \end{array}\right) \cdot N \cdot L\right)$ because DIG is applied for all combinations of class labels and sets of L attributes. This complexity depends on the number of instances, N, linearly and DIG can discover rules with high interestingness efficiently from large databases with many training instances. Instead, the complexity increases exponentially as the body length L increases and DIG can not discover rules with long bodies. However, simple rules with small number of attributes in their bodies are usually more important for analyzing databases than complex rules with many attributes because human experts can easily understand what the discovered rules mean.

3.5 Experiments

This section reports experimental results with five databases from UCI machine learning repository. This section shows how the restriction of rules with respect to the

Table 3.1: Databases used in the experiments. Domain size is the average number of possible values for nominal attributes.

name	# of classes	# of atts (nominals)	domain size	# of instances
dna	3	60(60)	4.0	3,175
krkp	2	36(36)	2.0	3,196
german	2	20(13)	4.3	1,000
mushroom	2	22(22)	5.7	8,124
promoters	2	57(57)	4.0	106

proposed interestingness affects the time complexity of DIG and the number of discovered rules. This section also reports examples of rules discovered by DIG.

Table 3.1 shows the characteristics of the databases used in the experiments. DIG can extract rules from instances represented with only nominal attributes. German database is represented by thirteen nominal and seven numeric attributes but only nominal attributes were used in the experiments. Other databases are represented by nominal attributes only. In all experiments, the lower bound of support, LB_{sup}, was set to 2% for all databases excluding promoters. Because promoters involves only 106 instances and 2% of them is too small, I set $LB_{sup} = 10\%$ for this database. The lower bound of accuracy was set to 90% for all databases in all experiments.

Table 3.2 and 3.3 show the running time of DIG and the number of discovered rules for various thresholds on the interestingness IG, LB_{int}. Table 3.2 is the result to extract rules with two conditions($L = 2$) and Table 3.3 is the result for rules with three conditions($L = 3$). In both tables, the running time of DIG is *not* the time for one DIG call for a specific combination of a target class and a set of L attributes, but is the total time for extracting rules for all class labels and for all sets of L attributes. The second column in the tables, # of combs, is the number of combinations of class labels and sets of L attributes, i.e. the number of DIG calls in each database.

Table 3.2: The number of discovered rules and the running time of DIG. The running time is the total cpu time to discover rules with two attributes ($L = 2$). This table shows the dependency on the lower bound of the interestingness. The lower bounds of support and accuracy are 2%(10% for promoters database only) and 90% respectively.

	# of combs	LB_{int}	# of rules	cpu time(sec)
dna	5,310	$-\infty$	145	155
		0.0	85	22
		0.1	17	20
		0.3	0	19
		0.5	0	20
		0.7	0	20
krkp	1,260	$-\infty$	82	7
		0.0	73	7
		0.1	10	7
		0.3	3	7
		0.5	3	7
		0.7	3	7
german	156	$-\infty$	31	95
		0.0	7	0.3
		0.1	3	0.3
		0.3	2	0.3
		0.5	0	0.3
		0.7	0	0.3
mushroom	460	$-\infty$	17,769	540
		0.0	4,899	11
		0.1	215	7
		0.3	105	7
		0.5	52	7
		0.7	15	7
promoters	3,192	$-\infty$	135	68
		0.0	61	1
		0.1	14	1
		0.3	2	1
		0.5	1	1
		0.7	0	1

Table 3.3: The number of discovered rules and the running time of DIG. The running time is the total cpu time to discover rules with three attributes ($L = 3$). This table shows the dependency on the lower bound of the interestingness. The lower bounds of support and accuracy are 2%(10% for promoters database only) and 90%.

	# of combs	LB_{int}	# of rules	cpu time(sec)
dna	102,660	$-\infty$	4,095,272	972,012
		0.0	508,444	11,114
		0.1	1,052	1,066
		0.3	1	938
		0.5	0	936
		0.7	0	936
krkp	14,280	$-\infty$	4,657	201
		0.0	4,334	174
		0.1	25	173
		0.3	2	173
		0.5	2	173
		0.7	0	173
german	572	$-\infty$	151,291	74,220
		0.0	25,976	11,124
		0.1	415	278
		0.3	31	51
		0.5	0	7
		0.7	0	3
mushroom	3,080	$-\infty$	-	$>$ 10 days
		0.0	9,713,290	236,238
		0.1	175	99
		0.3	26	98
		0.5	10	98
		0.7	1	99
promoters	58,520	$-\infty$	1,462,162	130,030
		0.0	815,513	25,708
		0.1	5,672	2,586
		0.3	489	767
		0.5	9	190
		0.7	0	90

The results in Table 3.2 and 3.3 show that the number of discovered rules strongly depended on the lower bound of the interestingness, LB_{int}. When $L = 3$ and LB_{int} was small, DIG discovered thousands or millions of rules. It is meaningless to show all of them to users of discovery systems and the restriction of rules with only support and accuracy was not sufficient to constrain rules to be discovered. However, most of the rules were rejected by low thresholds on the interestingness such as $LB_{int} = 0.1$ and $LB_{int} = 0.3$. This means that most of the rules with large support and high accuracy were statistically obvious rules because the rules with low scores of IG represent trivial relationships that can be easily expected from simpler rules.

In Table 3.2, the running time of DIG for rules with two conditions was independent of LB_{int} when $LB_{int} \geq 0.3$. The running time for $L = 3$ was also constant for high LB_{int} in databases excluding german and promoters. Instead, the running time increased rapidly with LB_{int} decreased in all databases except krkp. These results justify the discussion in the previous section that when LB_{int} is high, counting of class distributions is the most time consuming process and the complexity of this process is independent of LB_{int}. As LB_{int} decreases, more primitive rules satisfy minimum interestingness and DIG must to generate more rules with grouping corresponding to all subsets of the interesting primitive rules. Then, the process of generating rules with grouping increases rapidly as LB_{int} decreases and becomes more computationally heavy than counting class distribution. In krkp, the running time didn't increase for small LB_{int} at all. This is because most attributes in this database are binary and the number of rules with grouping is relatively small.

DIG was sufficiently fast when $L \leq 3$ and $LB_{int} \geq 0.3$ and required at most fifteen minutes for rules with three conditions from dna database that is the database with maximum number of attributes. For smaller LB_{int}, DIG required a few hours or a few

days and was too slow. However, the rules with such low score for the interestingness IG are usually useless from a statistical viewpoint and they are not required in usual discovery tasks.

Figure 3.5 shows examples of discovered rules from mushroom, promoters and krkp. For example, Rule 2 represents a rule for class *edible* with three conditions in mushroom database. Even if any condition in its body and any pair of conditions are mostly independent of or negatively relate to the conclusion, the combination of the three conditions concludes *edible* with more than 90% accuracy. In other words, all of the three conditions are required to classify instances into *edible* and the rule doesn't involve redundant conditions. Other rules represent similar relations in the corresponding databases. It is not clear whether these kinds of rules are really useful in real world applications of knowledge discovery, but they are quite interesting rules from statistical viewpoint.

3.6 Chapter Summary

This chapter discussed discovery of interesting rules for analyzing characteristics of databases represented by nominal attributes. In discovery for analysis, there is no clear measure to constrain rules to be discovered. This chapter first pointed out the limitation of rule filtering only with support and accuracy, and proposed a new criterion IG to evaluate classification rules. This criterion evaluates whether a rule to be evaluated gives us new information for its concluding class by comparing real accuracy of the rule and the expectation of accuracy calculated with its generalizations. In other words, This criterion evaluates a rule by whether all of the conditions in its body are really required to imply its conclusion with high accuracy.

This chapter also presented a new discovery algorithm DIG to extract interesting

Rule 1: $cap\_color \in \{brown, red\} \wedge popuration = solitary \Rightarrow edible$
$support = 8\%$, $accuracy = 100\%$, $interestingness = 0.79$
$P(edible) = 52\%$
$P(edible|cap\_color \in \{brown, red\}) = 50\%$, $P(edible|popuration = solitary) = 62\%$

Rule 2: $cap\_surface = smooth \wedge bruises = no \wedge gill\_color \in \{chocolate, pink\} \Rightarrow edible$
$support = 3\%$, $accuracy = 91\%$, $interestingness = 0.60$
$P(edible) = 52\%$
$P(edible|cap\_surface = smooth) = 45\%$, $P(edible|bruises = no) = 31\%$
$P(edible|gill\_color \in \{chocolate, pink\}) = 47\%$
$P(edible|cap\_surface = smooth \wedge bruises = no) = 43\%$
$P(edible|cap\_surface = smooth \wedge gill\_color \in \{chocolate, pink\}) = 51\%$
$P(edible|bruises = no \wedge gill\_color \in \{chocolate, pink\}) = 37\%$

(a) Discovered rules from mushroom

Rule 3: $a_{32} = G \wedge a_{49} \in \{C, T\} \Rightarrow promoter$
$support = 11\%$, $accuracy = 92\%$, $interestingness = 0.46$
$P(promoter) = 50\%$
$P(promoter|a_{32} = G) = 62\%$, $P(promoter|a_{49} \in \{C, T\}) = 56\%$

Rule 4: $a_6 \in \{A, C\} \wedge a_{29} \in \{T, G\} \wedge a_{38} \in \{T, G\} \Rightarrow \neg promoter$
$support = 11\%$, $accuracy = 92\%$, $interestingness = 0.60$
$P(\neg promoter) = 50\%$
$P(\neg promoter|a_6 \in \{A, C\}) = 52\%$, $P(\neg promoter|a_{29} \in \{T, G\}) = 49\%$
$P(\neg promoter|a_{38} \in \{T, G\}) = 50\%$
$P(\neg promoter|a_6 \in \{A, C\} \wedge a_{29} \in \{T, G\}) = 60\%$
$P(\neg promoter|a_6 \in \{A, C\} \wedge a_{38} \in \{T, G\}) = 61\%$
$P(\neg promoter|a_{29} \in \{T, G\} \wedge a_{29} \in \{T, G\}) = 60\%$

(b) Discovered rules from promoters

Rule 5: $bkxbq = f \wedge r2ar8 = t \wedge wkovl = f \Rightarrow nowin$
$support = 5\%$, $accuracy = 95\%$, $interestingness = 0.50$
$P(nowin) = 48\%$
$P(nowin|bkxbq = f) = 54\%$, $P(nowin|r2ar8 = t) = 53\%$, $P(nowin|wkovl = f) = 45\%$
$P(nowin|bkxbq = f \wedge r2ar8 = t) = 61\%$,$P(nowin|bkxbq = f \wedge wkovl = f) = 57\%$
$P(nowin|r2ar8 = t \wedge wkovl = f) = 61\%$

(c) Discovered rules from krkp

Figure 3.2: Examples of Discovered Rules.

rules with respect to the proposed criterion. To extract rules with grouping attribute values, this algorithm first enumerates interesting primitive rules without grouping and combines the discovered primitive rules to generate interesting rules with grouping. The empirical results showed DIG could extract rules with less than four conditions from databases consist of thousands of instances represented by dozens of nominal attributes.

There are some remaining limitations of DIG to be improved. First, the current DIG doesn't use thresholds of support and accuracy to restrict search space at all and only uses them for filtering generated rules with high interestingness. By using these thresholds, it is possible to accelerate DIG especially when the thresholds are relatively high. For example, when combining primitive rules for each subset S' of the set of all interesting primitive rules S at step 7 in Figure 3.1, DIG doesn't need to explore rules generated from any $S'' \subset S'$ if the rule from S' doesn't satisfy the constraint on minimum support. Oppositely, the number of negative instances covered by a rule generated from S' increases monotonically with S'. Then, it is possible to calculate an upper bound of accuracy of rules corresponding to $S''' \supset S'$ and DIG can prune all $S''' \supset S'$ if this upper bound is smaller than the given threshold of accuracy.

Another limitation is that DIG can only extract rules with the conditions on nominal attributes and can not deal with numeric attributes. The next chapter will discuss discovery from numeric databases.

Chapter 4

Discovery of Interesting Rules in Numeric Domains

4.1 Classification Rules in Numeric Domains

This chapter investigates discovery of classification rules from training instances represented by class labels and values of numeric attributes. There are many types of classification rules in numeric domains depending on what kind of conditions appear in their bodies[30, 27, 32]. This chapter focuses on the following type of rules.

$$x_1 \in [s_1, e_1] \wedge \cdots \wedge x_L \in [s_L, e_L] \Rightarrow class = c. \tag{4.1}$$

Each condition $x_i \in [s_i, e_i]$ in the body requires that a value of a_i must belong to the interval defined by boundaries s_i and e_i, and the body permits instances in the hyper-rectangle defined by the intervals for values of L attributes. This type of rules are used in many learning/discovery systems such as C4.5[51].

To evaluate classification rules, this chapter uses the interestingness presented in the previous chapter.

$$I(B \rightarrow c) = \frac{P(c|B) - \max_{1 \leq k \leq L} (P(c|B_k), \frac{P(c|x_k \in [s_k, e_k])P(c|B_k)}{P(c)})}{1 - P(c)}, \tag{4.2}$$

where B is a body of a rule to be evaluated and B_k is a set of conjunction of $L-1$ conditions excluding $x_k \in [s_k, e_k]$.

$$B_k = (x_1 \in [s_1, e_1] \wedge \cdots \wedge x_{k-1} \in [s_{k-1}, e_{k-1}] \wedge x_{k+1} \in [s_{k+1}, e_{k+1}] \wedge \cdots \wedge x_L \in [s_L, e_L]).$$

Because this definition of interestingness coincides with accuracy excluding a constant factor $\frac{1}{1-p(c)}$ for rules with one condition in their bodies, this chapter focuses on rules with two or more conditions. As described in the previous chapter, this interestingness requires that accuracy of an interesting rule should be much higher than expected from its generalization and all of the conditions in its body are required to imply its conclusion.

This chapter deals with a problem to enumerate classification rules like (4.1) whose support, accuracy and the interestingness equal to or higher than the given thresholds. For simplicity, this chapter assumes that values of each attribute are discretized into integers from 1 to D and that the number of attributes in the bodies, L, is given. Then the discovery of rules is a problem to select appropriate boundaries of intervals for each combination of a class label and L attributes.

4.2 Difficulty of Rule Discovery in Numeric Domains

One of the most serious difficulties to enumerate rules in numeric domains is that the number of possible rules is quite large. This difficulty becomes hard to resolve when D is large and values of each attribute are discretized into many values. Because the number of possible intervals for one attribute is $\frac{D(D+1)}{2}$, the number of possible rules with L attributes in their bodies is

$$K\binom{M}{L}\left(\frac{D(D+1)}{2}\right)^L,$$

where K is the number of class labels and M is the number of numeric attributes.

If the purpose of rule discovery is to find only typical rules with large support, it is not so important to decide precise boundaries of intervals because the typical rules cover many instances and small differences of boundaries cause only slight effects on their support, accuracy and the interestingness. In such a case, rough discretization with small D is sufficient for rule discovery and it is not difficult to generate all of possible rules and to evaluate their support, accuracy and interestingness.

Oppositely, accuracy and the interestingness of rules with small support may change significantly with small differences of boundaries of intervals. There may be interesting rules with small support as discussed in the previous chapter and we need to discretize each attribute into relatively many values to discover such rules. This increases the number of possible rules significantly. For example, when D=10 and values of every attribute are discretized into 10 values, the number of rules with three attributes in their bodies is about 10^5 for a fixed combination of a class label and the three attributes in the bodies. However, the number of rules increases to more than 10^9 for D=50. Discovery algorithms have to explore all combinations of class labels and L attributes, and it is not practical to generate and evaluate all of possible rules. The key issue of rule discovery in numeric domains is an efficient search technique to find desired rules from huge number of possible rules.

4.3 Discovery Algorithms

This section presents two discovery algorithms in numeric domains. To simplify discussion, these algorithms requires a class label and a set of attributes appear in their bodies as their inputs. Therefore, the inputs of the algorithms are a class label, a set of L attributes, and thresholds for support, accuracy and the interestingness. The discovery algorithms output a set of rules whose support, accuracy and the interestingness are higher than or equal to the given thresholds. To discover all rules, the algorithms are applied to all combinations of class labels and sets of L attributes.

Both of two algorithms in this section are based on tree search technique that starts from a root node covering all of possible rules and continues to divide the rules gradually at each internal node in search tree. Tree search algorithms are characterized by how to divide rules at internal nodes and how to evaluate upper bounds of the criteria of rules to prune them at internal nodes. The two algorithms in this section differ from each other in dividing methods. One of them divides rules by fixing a start or a finish point of an interval for a certain attribute and the other divides rules by reducing a range of a start or a finish point.

The efficiency of the discovery algorithms strongly depends on the ability to prune internal nodes. If the algorithm can prune internal nodes with many rules near a root node, then the algorithm can enumerate all interesting rules in practical time even if there are huge number of possible rules. The two algorithms basically use the same pruning strategy. They evaluate upper bounds of support, accuracy and interestingness of the rules corresponding to each internal node with the same evaluation functions, and prune the rules if at least one evaluated upper bound becomes smaller than the corresponding threshold.

4.3.1 SAB: Rule Discovery by Selecting Attribute Boundaries

Figure 4.1 shows a pseudo-code of a rule discovery algorithm SAB(Select Attribute Boundaries) that divides rules by selecting a start and finish point of an interval for one attribute. It is a standard method used in many learning algorithms such as C4.5 to select a boundary for each attribute one by one to learn classification rules in numeric domains.

procedure SAB($h, s_1, e_1, \ldots, s_h, e_h$)
inputs:
h : the number of attributes whose intervals are fixed.
s_i : selected start point of an interval for x_i $(i = 1, \ldots, h)$.
e_i : selected end point of an interval for x_i $(i = 1, \ldots, h)$.
output:
Rules : a set of discovered rules.
1. Calculate upper bounds of support, f_{sup}, accuracy, f_{acc} and interestingness f_{int}.
2. **If** $f_{sup} < LB_{sup}$ or $f_{acc} < LB_{acc}$ or $f_{int} < LB_{int}$ **then**
3. $Rules := \phi$.
4. **else**
5. **If** $h = L$ **then**
6. $Rules := \{x_1 \in [s_1, e_1] \wedge \cdots \wedge x_L \in [s_L, e_L] \Rightarrow class = c\}$.
7. **else**
8. $Rules := \phi$.
9. **for**(s_{h+1}:=1;$s_{h+1} \leq D \wedge f_{sup}(s_1, e_1, \ldots, s_h, e_h, s_{h+1}, D) \geq LB_{sup}$; s_{h+1}++)
10. **for**(e_{h+1}:=D; $e \geq 1 \wedge f_{sup}(s_1, e_1, \ldots, s_{h+1}, e_{h+1}) \geq LB_{sup}$; e_{h+1} −)
11. $Rules := Rules \cup SAB(h + 1, s_1, e_1, \ldots, s_{h+1}, e_{h+1})$.
10. **Return** *Rules*.

Figure 4.1: Discovery algorithm SAB that searchs for classfication rules by selecting boundaries of intervals of attributes in their bodies. For simplicity, this figure ignores thresholds of support, accuracy and interesgingness, LB_{sup}, LB_{acc} and LB_{int} as the arguments(inputs) of SAB.

SAB is a recursive procedure and is first called with $h = 0$, i.e. no boundary is selected. When called with selected intervals for attributes, $x_1, \ldots, x_h$, SAB first calculates upper bounds of support, accuracy and the interestingness of rules with selected intervals. If at least one of the upper bound becomes smaller than the corresponding threshold, then SAB returns empty set as a set of discovered rules(step 3). Otherwise, if $h = L$ and all intervals are fixed, then SAB completely instanciates the rule that satisfies the requirements on support, accuracy and the interestingness. Then, SAB returns the rule as an interesting rule (step 6). If $h < L$, SAB selects a boundary of x_{h+1}, s_{h+1} and e_{h+1}, and calles SAB recursivelly with the selected boundary. Because

support of rules monotonically decreases with s_{h+1} increases and e_{h+1} decreases, SAB stops for-loops in step 9 and step 10 when the upper bound of support, f_{sup}, becomes lower than the given threshold for support, LB_{sup}.

The performance of discovery algorithm strongly depends on how to evaluate upper bounds of support, accuracy and the interestingness. First, because of the following trivial relationship

$$\begin{aligned} & P(c \wedge x_1 \in [s_1, e_1] \wedge \cdots \wedge x_L \in [s_L, e_L]) \\ \leq\ & P(c \wedge x_1 \in [s_1, e_1] \wedge \cdots \wedge x_L \in [s_{L-1}, e_{L-1}]) \\ \leq\ & P(c \wedge x_1 \in [s_1, e_1] \wedge \cdots \wedge x_L \in [s_{L-2}, e_{L-2}]) \\ \leq\ & \cdots, \end{aligned}$$

the upper bound of support, f_{sup}, can be calculated as follows when $s_1, e_1, \ldots, s_h, e_h$ are fixed.

$$f_{sup}(s_1, e_1, \ldots, s_h, e_h) = P(c \wedge x_1 \in [s_1, e_1] \wedge \cdots \wedge x_h \in [s_h, e_h]). \tag{4.3}$$

This upper bound coincides with support itself when $h = L$, i.e. when the boundaries of all intervals are selected.

In contrast to the upper bound of support, it is not easy to evaluate the upper bound of accuracy. If boundaries for one or more attributes are not selected, then accuracy may become very low or very high depending on them and it is difficult to reduce the range of accuracy with practical computational complexity. Then, SAB uses the following trivial upper bound that becomes real accuracy when all boundaries are selected and becomes 1 otherwise.

$$f_{acc} = \begin{cases} P(c | x_1 \in [s_1, e_1] \wedge \cdots \wedge x_L \in [s_L, e_L]) & h = L \\ 1 & h < L \end{cases} \tag{4.4}$$

The difficulty of evaluating a range of accuracy affects the evaluation of the upper bound of the interestingness because the interestingness $I(B \Rightarrow c)$ at (4.2) depends

on both of accuracy of the rule, $P(c|B)$, and accuracy of more general rules, $P(c|x_k \in [s_k, e_k])$ and $P(c|B_k)$. Because $I(B \Rightarrow c)$ positively depends on $P(c|B)$ and negatively depends on $P(c|x_k \in [s_k, e_k])$ and $P(c|B_k)$, its upper bound can be estimated by replacing $P(c|B)$ with its upper bound, f_{acc} in (4.4) and replacing $P(c|x_k \in [s_k, e_k])$ and $P(c|B_k)$ with their lower bounds. As similar to the upper bound of accuracy, SAB uses 0 for the lower bound of accuracy $P(c|B_k)$ if it involves a condition on an attribute whose interval is not selected. Similarly, the lower bound of $P(c|x_k \in [s_k, e_k])$ is 0 if s_k and e_k are not selected. In all boundaries are fixed, SAB can count the probabilities as their lower bound. When $h < L$, at least one of B_k or $x_k \in [s_k, e_k]$ involves unselected intervals and we can ignore the second term in max in (4.2). As a result, SAB applies the following upper bound of interestingness when intervals for $x_1, \ldots, x_h$ are selected.

$$f_{int} = \begin{cases} I\left(x_1 \in [s_1, e_1] \wedge \cdots \wedge x_L \in [s_L, e_L] \Rightarrow class = c\right) & h = L \\ \frac{1-P(c|B_L)}{1-P(c)} & h = L-1 \\ \frac{1}{1-P(c)} & h \leq L-2 \end{cases} \tag{4.5}$$

4.3.2 IIS: Rule Discovery by Iterative Interval Specialization

The second discovery algorithm, IIS(Iterative Interval Specialization), applies a different dividing approach. IIS also divides rules based on a value of s_i and e_i as SAB does. However IIS doesn't fix a value of s_i or e_i at each internal node but reduces ranges of them.

Figure 4.2 shows a pseudo-code of IIS that takes lower and upper bounds of a start and an end point of an interval for each attribute, $s_i^-, s_i^+, e_i^-, e_i^+$, as its inputs and searches for rules such that

$$x_1 \in [s_1, e_1] \wedge \cdots \wedge x_L \in [s_L, e_L] \Rightarrow class = c,$$

procedure IIS($s_1^-, s_1^+, e_1^-, e_1^+, \ldots, s_L^-, s_L^+, e_L^-, e_L^+$)
inputs:
s_i^-, s_i^+ : lower and upper bounds of $s_i (i = 1, \ldots, L)$.
e_i^-, e_i^+ : lower and upper bounds of $e_i (i = 1, \ldots, L)$.
output:
Rules : a set of discovered rules.
1. Calculate upper bounds of support, g_{sup}, accuracy, g_{acc} and interestingness g_{int}.
2. **If** $g_{sup} < LB_{sup}$ or $g_{acc} < LB_{acc}$ or $g_{int} < LB_{int}$ **then**
3. $Rules := \phi$
4. **else**
5. **If** $s_i^- = s_i^+$ and $e_i^- = e_i^+$ for all $1 \leq i \leq L$ **then**
6. $Rules := \left\{x_1 \in [s_1^-, e_1^-] \wedge \cdots \wedge x_L \in [s_L^-, e_L^-)] \Rightarrow class = c\right\}$.
7. **else**
8. Select s_i or e_i whose lower bound and upper bound are different.
9. **If** s_h is selected **then**
10. $u_{new} = \lfloor \frac{s_h^- + s_h^+}{2} \rfloor$
11. $Rules := \text{IIS}(s_1^-, s_1^+, e_1^-, e_1^+, \ldots, s_h^-, u_{new}, e_h^-, e_h^+, \ldots, s_L^-, s_L^+, e_L^-, e_L^+)$ $\cup \text{IIS}(s_1^-, s_1^+, e_1^-, e_1^+, \ldots, u_{new}+1, s_h^+, e_h^-, e_h^+, \ldots, s_L^-, s_L^+, e_L^-, e_L^+)$
12. **else** (e_h is selected)
13. $u_{new} = \lfloor \frac{e_h^- + e_h^+}{2} \rfloor$
14. $Rules := \text{IIS}(s_1^-, s_1^+, e_1^-, e_1^+, \ldots, s_h^-, s_h^+, e_h^-, u_{new}, \ldots, s_L^-, s_L^+, e_L^-, e_L^+)$ $\cup \text{IIS}(s_1^-, s_1^+, e_1^-, e_1^+, \ldots, s_h^-, s_h^+, u_{new}+1, e_h^+, \ldots, s_L^-, s_L^+, e_L^-, e_L^+)$
15. **Return** *Rules*.

Figure 4.2: Discovery algorithm IIS.

where

$$s_i^- \leq s_i \leq s_i^+, e_i^- \leq e_i \leq e_i^+, 1 \leq \forall i \leq L.$$

IIS is also a recursive procedure and is first called with the ranges $s_i^- = 1, s_i^+ = D, e_i^- = 1, e_i^+ = D (1 \leq i \leq L)$ that cover all of possible rules. IIS first calculates upper bounds of support, accuracy and interestingness of the rules that satisfy the ranges of boundaries given as inputs (step 1). If the upper bounds become smaller than the given thresholds, then there is no rule to be discovered in the ranges of boundaries and

IIS returns an empty set (step 3). Otherwise, if $s_i^- = s_i^+$ and $e_i^- = e_i^+$ for all i, i.e. if values of all start points and end points are fixed, then IIS returns the rule defined by the boundary points because step 2 guarantees that the rule satisfies the requirements on its support, accuracy and the interestingness. If there is a start point s_h or an end point e_h whose range permits plural values, IIS divides the current range of s_h (e_h) into two disjoint ranges at the middle point, u_{new}, between s_h^- (e_h^-) and s_h^+ (e_h^+). IIS calls IIS recursively for each new range and returns the union of the results of the recursion calls.

The time complexity of IIS strongly depends on how to calculate the upper bounds of support, accuracy, and the interestingness at step 1, and how IIS effectively prunes inadequate rules with the upper bounds. Let $I_i^+ = [s_i^-, e_i^+]$ and $I_i^- = [s_i^+, e_i^-]$ respectively. Because $I_i^- \subseteq [s_i, e_i] \subseteq I_i^+$, the probability of positive instances with class=c covered by the rule

$$x_1 \in [s_1, e_1] \wedge \cdots \wedge x_L \in [s_L, e_L] \Rightarrow class = c$$

is at most $P(c \wedge x_1 \in I_1^+ \wedge \ldots \wedge x_L \in I_L^+)$ and the probability of negative instances is at least $P(\neg c \wedge x_1 \in I_1^- \wedge \ldots \wedge x_L \in I_L^-)$. This leads the following upper bound of support,

$$g_{sup} = P(c \wedge x_1 \in I_1^+ \wedge \ldots \wedge x_L \in I_L^+), \tag{4.6}$$

and the upper bound of accuracy,

$$g_{acc} = \frac{P(c \wedge x_1 \in I_1^+ \wedge \ldots \wedge x_L \in I_L^+)}{P(c \wedge x_1 \in I_1^+ \wedge \ldots \wedge x_L \in I_L^+) + P(\neg c \wedge x_1 \in I_1^- \wedge \ldots \wedge x_L \in I_L^-)}, \tag{4.7}$$

because accuracy of a rule becomes maximum when it covers as many positive instances as possible and as less negative instances as possible.

As discussed in subsection 4.3.1, we need lower bounds of accuracy of general rules, $x_k \in [s_k, e_k] \Rightarrow c$ and $B_k \Rightarrow c$, to evaluate the upper bound of the interestingness, g_{int}.

By similar way with evaluation of g_{acc}, we can evaluate a lower bound of accuracy, g_{acc}^{lb} of a rule with less than L conditions in its body,

$$x_{i_1} \in [s_{i_1}, e_{i_1}] \wedge \cdots \wedge x_{i_k} \in [s_{i_k}, e_{i_k}] \Rightarrow class = c, 1 \leq i_1 < \ldots < i_k \leq L, k < L$$

as follows.

$$\begin{aligned} & g_{acc}^{(lb)}(x_{i_1} \in [s_{i_1}, e_{i_1}] \wedge \cdots \wedge x_{i_k} \in [s_{i_k}, e_{i_k}] \Rightarrow class = c) \\ = & \frac{P(c \wedge x_{i_1} \in I_{i_1}^- \wedge \ldots \wedge x_{i_k} \in I_{i_k}^-)}{P(c \wedge x_{i_1} \in I_{i_1}^- \wedge \ldots \wedge x_{i_k} \in I_{i_k}^-) + P(\neg c \wedge x_{i_1} \in I_{i_1}^+ \wedge \ldots \wedge x_{i_k} \in I_{i_k}^+)}. \end{aligned} \tag{4.8}$$

With this lower bound we can calculate the upper bound of interestingness, g_{int}, as follows.

$$g_{int} = \frac{g_{acc} - \max\limits_{1 \leq k \leq L} \left(g_{acc}^{(lb)}(B_k \Rightarrow c), \frac{g_{acc}^{(lb)}(x_k \in [s_k, e_k] \Rightarrow c) g_{acc}^{(lb)}(B_k \Rightarrow c)}{P(c)} \right)}{1 - P(c)} \tag{4.9}$$

It is worth to note that the upper bounds (4.6)$\sim$(4.9) for IIS coincide with the upper bounds for SAB defined by (4.3)$\sim$(4.5) when $s_i^- = s_i^+$ and $e_i^- = s_i^+$ for $i \leq h$ and $s_i^- = e_i^- = 1$ and $s_i^+ = e_i^+ = D$ for $i > h$, i.e. when the boundaries are fixed for $x_i, i \leq h$ and are not restricted at all for $x_i, i > h$ as SAB does. In other words, SAB and IIS applies the same evaluation functions for pruning. The difference of SAB and IIS is how to divide rules at each internal node of a search tree. SAB divides rules by values of a start/end point of an interval for a certain attribute and generates a shallow but wide search tree. IIS divides rules by whether a certain start/end point is higher than a specific value. IIS divides rules into two sets at each internal node and generates a deep search tree. As the experiments in the next section shows, this difference causes significant difference of their time complexities.

At the end, both of SAB and IIS can enumerate all rules whose support, accuracy and the interestingness are higher than or equal to the corresponding thresholds and are *not* approximation method that may miss some rules.

Table 4.1: Databases used in the experiments. All attributes in these databases are numeric and only instances with no unknown attribute value were used in the experiments.

name	# of classes	# of attributes	# of instances
satellite	6	36	6,435
segment	7	19	2,310
vehicle	4	18	846
waveform	3	21	5,000

4.4 Experiments

This section reports experimental results for time complexities of SAB and IIS, and shows what kind of rules are discovered by them. Table 4.1 shows the four databases form UCI repository[6] used in the experiments.

All attributes in these databases are numeric attributes. Some databases involves instances with missing attribute values but only the instances without missing value were used in the experiments in this section. The number of instances in Table 4.1 is not the size of original databases, but the number of remaining instances without unknown attribute value. The two algorithms discretized ranges of each attribute into values with same population.

4.4.1 Time Complexity

This subsection explores how the parameters of rule discovery such as D and L affect the time complexities of discovery algorithms. All experiments in this section were executed on a Sun workstation with 300MHz UltraSPARC-II.

Table 4.2 shows the dependency of running time of SAB and IIS on the number of discretized values. In this section, "running time" means the total time to discover rules from all combinations of class labels and attributes, and *not* for a certain pair of

Table 4.2: Dependency of the time complexity on D (the number of attribute values after discretization). The complexity is the total cpu time to discover rules with two attributes ($L = 2$) and whose support, accuracy and interestingness are higher than or equal to 0.02, 0.9 and 1.0 respectively. This table also shows the number of combinations of class labels and sets of two attributes.

	# of combs	D	SAB(sec)	IIS(sec)	SAB / IIS
satellite	3,780	8	55	47	1.2
		16	184	60	3.1
		32	1,544	89	17.3
		64	9,671	158	61.3
segment	1,197	8	6	6	1.2
		16	24	8	2.9
		32	196	15	13.1
		64	1,886	35	33.3
vehicle	612	8	4	3	1.6
		16	35	6	5.5
		32	241	9	26.1
		64	1,179	17	69.7
waveform	630	8	11	8	1.3
		16	65	14	4.8
		32	778	20	38.1
		64	2,018	37	54.9

a class label and L attributes. In Table 4.2, "# of combinations" shows the number of combinations of class labels and L attributes for each database. In the experiments for Table 4.2, I set L to 2 and set $LB_{sup}, LB_{acc}, LB_{int}$ to 0.02, 0.9 and 1.0 respectively. Then the running time in Table 4.2 is the time to discover all rules with two attributes in their bodies and whose support, accuracy and interestingness are higher than or equal to 2%, 90% and 1.0. From Table 4.2, we can observe that both of IIS and SAB could discover interesting rules in a few minutes or less and were sufficiently fast with rough discretization with $D = 8$ and $D = 16$. However, IIS was much faster than SAB when D was large and each attribute was discretized into many values. For example, IIS was $30 \sim 70$ times faster than SAB when D=64. As discussed at the

Table 4.3: Dependency of the time complexities of SAB and IIS on the number of conditions, L. Each attribute values were discretized into 16 values and the thresholds of support, accuracy and interestingness were 0.02, 0.9 and 0.6 respectively.

	L	# of combs	SAB(sec)	IIS(sec)	SAB / IIS
satellite	2	3,780	185	71	2.6
	3	42,840	196,613	17,653	11.1
segment	2	1,197	24	9	2.6
	3	6,783	7,695	864	8.9
vehicle	2	612	33	6	5.0
	3	3,020	15,863	894	17.7
waveform	2	630	65	19	3.5
	3	3,990	80,791	6,530	12.4

beginning of this chapter, there may be interesting rules with small support and we need to discretize attribute values into many values to discover such rules. The results in Table 4.2 shows IIS is a better choice than SAB to discover such rules effectively.

Table 4.3 shows the dependency of execution time on the number of conditions, L. Both of SAB and IIS required much more time to discover rules with three conditions than to discover rules with two conditions because the number of possible rules rapidly increases as the number of conditions, L, increases. For example, the number of possible rules with two conditions for satellite database is about 10^7 but the number of rules becomes about 10^{10} when $L = 3$. For this database, SAB required more than two days and it was too slow to apply to the usual knowledge discovery process. IIS also required about five hours and was more than ten times faster than SAB. Similar results are observed in other three databases that IIS was not so fast but was much efficient than SAB and the difference between SAB and IIS became larger as the number of conditions increases.

Table 4.4, 4.5 and 4.6 show the dependency on the thresholds for support, accuracy and interestingness respectively. An important result from these tables is that

Table 4.4: Dependency of the time complexity on the threshold for support, LB_{sup}. This table shows the results for $L = 2$, $D = 32$, $LB_{acc} = 0.9$ and $LB_{int} = 1.0$.

	LB_{sup}	SAB(sec)	IIS(sec)	SAB / IIS
satellite	0.01	2,020	111	18.3
	0.02	1,545	89	17.3
	0.05	865	70	12.4
segment	0.01	268	20	13.3
	0.02	197	15	13.1
	0.05	84	11	7.8
vehicle	0.01	288	15	19.3
	0.02	241	9	26.1
	0.05	139	4	32.8
waveform	0.01	971	39	24.9
	0.02	778	20	38.1
	0.05	448	11	42.3

Table 4.5: Dependency of the time complexity on the threshold for accuracy, LB_{acc}. This table shows the results for $L = 2$, $D = 32$, $LB_{sup} = 0.02$ and $LB_{int} = 0.5$.

	LB_{acc}	SAB(sec)	IIS(sec)	SAB / IIS
satellite	0.5	1,888	511	3.7
	0.7	1,685	327	5.2
	0.9	1,564	155	10.1
segment	0.5	211	54	3.9
	0.7	201	37	5.5
	0.9	196	21	9.3
vehicle	0.5	251	46	5.4
	0.7	242	24	10.2
	0.9	240	13	18.1
waveform	0.5	927	128	7.2
	0.7	800	81	9.9
	0.9	778	37	21.8

Table 4.6: Dependency of the time complexity on the threshold for interestingness, LB_{int}. This table shows the results for $L = 2$, $D = 32$, $LB_{sup} = 0.02$ and $LB_{acc} = 0.9$.

	LB_{int}	SAB(sec)	IIS(sec)	SAB / IIS
satellite	0.5	1,564	155	10.1
	1.0	1,544	89	17.3
	1.1	1,548	68	22.8
segment	0.5	196	21	9.3
	1.0	196	15	13.1
	1.1	194	11	17.1
vehicle	0.5	240	13	18.1
	1.0	241	9	26.1
	1.1	240	7	33.1
waveform	0.5	778	36	21.8
	1.0	778	20	38.1
	1.1	777	17	47.1

the execution time of SAB is almost independent of the thresholds for accuracy and interestingness. In general, higher threshold means more strict restriction on the rules to be discovered. Discovery algorithms can prune rules more efficiently with the higher and more strict threshold and can enumerate rules with shorter running time. The independence of SAB from the thresholds on accuracy and the interestingness means that SAB could not use them to prune inappropriate rules at all. In contrast, IIS became faster as the thresholds of accuracy and interestingness increased. This shows that IIS used these thresholds to prune rules effectively even if IIS and SAB used the same evaluation functions of accuracy and interestingness of rules. For the threshold of support, both of SAB and IIS improved their efficiency for high thresholds and the evaluation function of support worked well for pruning rules in both algorithms.

The experiments in this subsection show that IIS is faster than SAB in all cases. IIS can use thresholds of accuracy and interestingness effectively and becomes much efficient than SAB when discovering longer rules from instances represented by numeric

Table 4.7: Dependency of the number of rules on the threshold for interestingness, LB_{int}. This table shows the results for $L = 2$, $D = 32$, $LB_{sup} = 0.02$ and $LB_{acc} = 0.9$.

LB_{acc}	satellite	segment	vehicle	waveform
$-\infty$	1,974,150	255,344	49,561	19,532
0.1	938,358	92,334	36,914	17,268
0.2	869,825	85,687	33,975	14,303
0.3	803,451	78,477	32,054	11,090
0.4	739,093	71,780	28,826	7,681
0.5	683,613	65,721	23,813	5,017
0.6	615,531	57,933	17,219	1,918
0.7	518,556	49,225	10,076	294
0.8	336,703	39,340	4,874	47
0.9	147,974	28,097	1,921	0
1.0	36,128	13,050	398	0
1.1	300	1,976	25	0
1.2	0	0	1	0
1.3	0	0	0	0

attributes with many discretized values. IIS is sufficiently fast to discover rules with less than four conditions in their bodies from thousands of instances represented by dozens of attributes.

4.4.2 Discovered Rules

Table 4.7 shows how the threshold of the interestingness affects the number of discovered rules. Without any restriction on the interestingness, discovery algorithms output huge number of rules with typical setting of thresholds of support and accuracy, LB_{sup}=2% and LB_{acc}=90%. For example, there are millions of rules for satellite database that cover more than 2% of instances and classify them with 90% or higher accuracy. The restriction on the interestingness can filter out most of them and selects thousands or less rules with an appropriate threshold, $0.5 \sim 1.1$ for the databases in Table 4.7. The problem is whether the selected rules with high interestingness are

really interesting and useful.

Figure 4.3 shows examples of rules discovered by SAB and IIS. Figure 4.3 also shows probaibilties of concluding classes and accuracy of more general rules used to evaluate their interestingness. Both of SAB and IIS can enumerate all rules with given requirements on support, accuracy and the interestingness. Then, they output the same rule set for each database and the rules in 4.3 are dicscovered by both algorithms. Figure 4.3 shows two rules for each database discovered with $L = 2, D = 64, LB_{sup} = 0.02, LB_{acc} = 0.9$ and $LB_{int} = 0.8$. One is the rule with the highest interestingness and the other is the rule with the maximum support. Figure 4.3 doesn't show rules from waveform because no rule satisfied the given requirements.

Satellite database represents observations of geographic features from Landsat. In the database, values of attributes represent the intensities of four spectrum bands, two visible and two infra-red, of 3x3 pixels and a class label is a geographic class of center pixel such as *red_soil* and *damp_grey_soil*. Two attributes, a_{17} and a_{18}, in rule 1 are the intencities of two visible spectrum bands in the center pixel. Even if the conditions of these two visible spectrums have very small relationship with the conclusion, *red_soil*, but the combination of them implies *red_soil* with 100% accuracy. Rule 2 represents a similar relationship of the same combination of visible spectrums at the differenct pixcel. There exist similar rules for other pixels and the relationship of these two visible spectrums is independent of pixcels. It is not clear that these rules are realy useful for domain experts, but the rules are interesting from statistical view point.

4.5 Chapter Summary

This chapter investigated discovery of classification rules in numeric domains and presented two discovery algorithms to this task. The first algorithm, SAB, applies a

rule 1: $a_{17} \in [46, 51] \wedge a_{18} \in [64, 87] \rightarrow red\_soil$
$support = 0.02, accuracy = 1.00, interestingness = 0.93$
$P(red\_soil) = 0.238$
$P(red\_soil|a_{17} \in [46, 51]) = 0.288$, $P(red\_soil|a_{18} \in [64, 87]) = 0.216$

rule 2: $a_{33} \in [46, 56] \wedge a_{34} \in [67, 92] \rightarrow red\_soil$
$support = 0.06, accuracy = 0.99, interestingness = 0.80$
$P(red\_soil) = 0.238$
$P(red\_soil|a_{33} \in [46, 56]) = 0.379$, $P(red\_soil|a_{34} \in [67, 92]) = 0.235$

(a)Discovered rules from satellite

rule 3: $distance\_circularity \in [79, 91]$
$\wedge scaled\_variance\_minor\_axis \in [519, 686] \rightarrow saab$
$support = 0.02, accuracy = 0.95, interestingness = 1.07$
$P(saab) = 0.258$, $P(saab|distance\_circularity \in [79, 91]) = 0.153$
$P(saab|scaled\_variance\_minor\_axis \in [519, 686]) = 0.145$

rule 4: $max\_length\_aspect\_ratio \in [9, 12]$
$\wedge scaled\_variance\_minor\_axis \in [304, 367] \rightarrow bus$
$support = 0.08, accuracy = 0.99, interestingness = 0.80$
$P(van) = 0.235$, $P(van|max\_length\_aspect\_ratio \in [9, 12]) = 0.288$
$P(van|scaled\_variance\_minor\_axis \in [304, 367]) = 0.305$

(b)Discovered rules from vehicle

rule 5: $exred\_mean \in [-7.7, -2.8]$
$\wedge exblue\_mean \in [19.7, 27.8] \rightarrow brickface$
$support = 0.02, accuracy = 0.98, interestingness = 0.92$
$P(brickface) = 0.143$
$P(brickface|exred\_mean \in [-7.7, -2.8]) = 0.147$
$P(brickface|exblue\_mean \in [19.7, 27.8]) = 0.182$

rule 6: $rawblue\_mean \in [0.8, 10.5]$
$\wedge hue\_mean \in [-2.094303, -1.563972] \rightarrow window$
$support = 0.03, accuracy = 0.91, interestingness = 0.80$
$P(window) = 0.143$
$P(window|rawblue\_mean \in [0.8, 10.5]) = 0.198$
$P(window|exblue\_mean \in [-2.094303, -1.563972]) = 0.159$

(c)Discovered rules from segment

Figure 4.3: Examples of discovered rules.

traditional approach in rule discovery in numeric domains that iteratively determines a boundary of possible values for each attribute one by one. The second algorithm, IIS, doesn't fix a boundary at each step. Instead, it iteratively reduces a range of a boundary at each step. The experimental results showed that IIS is much faster than SAB in many situations even if the two algorithms apply completely the same evaluation functions for pruning rules.

There are two reasons why IIS is much efficient than SAB. First, SAB instanciates a condition on each attribute one by one and gives no restriction on the condition of the last attribute until just before leaf nodes. This makes it difficut to evaluate upper bounds of accuracy and the interestingness accurately even if SAB uses the same upper bound functions with IIS. Second, SAB divides focused rules into many subsets with small number of rules at each decision point by selecting values of a start or an finish point of an interval. As a result, even if SAB prunes internal nodes near a root node of a search tree, SAB can reject only a small number of rules. Instead, IIS divides rules into only two sets at each decision point and can reject many rules if pruning occures near the root.

One of the remanining problems of IIS is that it outputs many quite similar rules. The rules share a same concluding class and their bodies consist of similar conditions on the same attributes. As a result, the hyper-rectangles defined by their bodies almost concide with each other and the rules share many instances covered by their bodies. Such rules represent an identical relationship in the given instances and it is not good to show all of them to a user of discovery system because it degrades readability of discovered rules.

Chapter 5

Related Works

5.1 Learning Decision Rules

Chapter 2 discussed learning hypotheses from a set of pre-classified instances to classify new instances correctly. This task is known as supervised learning and is one of the most typical framework of machine learning. There are many types of representations of hypotheses and this thesis used a set of classification rules. Classification rules are also called decision rules in the field of supervised learning.

Michalski's AQ family [38, 40, 68] is one of the most basic researches for learning decision rules. AQ starts from a set of all training instances and learns rules one by one. It first selects a positive instance randomly as a seed and learns a rule for the selected instance. After that, it selects another instance not covered by the learned rule as a new seed and learns another rule. AQ iterates this process until all of positive instances are covered by at least one rule. Many other learning algorithms such as CN2[14, 15] and RIPPER[17] also learn decision rules one by one. However, they differ from AQ in how to learn each rule. As discrebed above, AQ selects a seed instance and learns a rule for the instance. Instead, CN2 and RIPPER don't select a seed instance but try to learn a rule that works best for the training instances that are not covered by the learned rules. The learning algorithm proposed in Chapter 2, IGR, uses a seed instance

to learn each rule and is similar to AQ. The most important difference between AQ and IGR is that IGR always induces a rule for every positive instance even if some of them are redundant to cover positive instances. Oppositely, AQ stops learning when all positive instances are covered. As a result, IGR learns more rules than AQ. Another difference is that IGR induces all rules simultaneously to improve learning time.

Learning decision rules consists of two phases, growing rules and pruning the grown rules. Growing phase is more important for classification ability while pruning phase is useful to avoid over fitting on the training instances and to improve classification performance. Reduced error pruning[16] is the most widely used method for pruning decision rules. It starts from the original decision rules from growing phase and iteratively simplifies it with the simplification operator with the maximum gain of classification error for new instances estimated with the training instances. In usual, reduced error pruning uses two simplification operators, removing a condition from a decision rule and removing a rule itself. The former operator generalizes the hypothesis and the later one specializes the hypothesis. Reduced error pruning is effective to improve classification performance in noisy databases but it requires much more time than growing phase [16] because it is not computationally easy to evaluate the effects of removing one condition of a certain rule[16]. IREP[28] resolved this problem by pruning each rule just after the rule is learned and before generating the next rule. Many empirical researches [16, 50] show that pruning learned hypotheses not only simplify the rules but also improve classification accuracy for new, unknown instances. The current IGR doesn't apply pruning and induces rules with 100% accuracy for training instances. As the experiments in Chapter 2 show, IGR achieved high classification accuracy and the pruning rules may not improve its accuracy significantly. However, from the viewpoint of simplicity of learned hypotheses, pruning rules will be able to

improve DIG.

Most learning algorithms including IGR apply general-to-special rule induction that starts from the most general rule and iteratively specializes it to increase its accuracy. This approach achieves good rules in many natural domains but sometimes leads poor rules. In each specialization step, general-to-special approach evaluates relationship between class and each attribute independently and selects the attribute with the highest relationship. Then, the learning algorithms sometimes miss hidden relationships such as exclusive-or of two attributes because each attribute has no or small relation to class labels by itself and the condition selection criteria give small scores to the attributes. There is another approach to rule induction, special-to-general approach. This approach starts from a rule corresponding to a specific training instance and iteratively generalizes it by removing conditions of attributes. This approach is more robust than general-to-special approach on the above hidden relationships because the accuracy of the rule decreases if any conditions on relevant attributes are removed. There is only a few learning algorithms based on this approach because special-to-general learning is computationally heavier than general to special learning. For example, RISE[20] starts from a rule set in which each rule corresponds to each training instance and constrains values of all attributes to permit only the corresponding instance. RISE generalizes the rule set by iteratively merging most similar rule pairs. Then, its time complexity is $O(N^2)$ or $O(N^3)$ depending on the number of resulting rules where N is the number of training instances.

IGR classifies a new instance with majority voting when the instance satisfies plural rules. Decision list[15, 56] is another approach to classify instances with decision rules. It orders the induced rules and uses only the first rule that matches the instance to be classified. In decision lists, it is easy to understand the logical meaning of the whole

hypothesis because the decision lists represent a chain of "if-then-else". In unordered sets of rules such as IGR induces, the meaning of the whole hypothesis is not clear because the boundaries between classes are defined by majority voting of plural rules. Instead, the meaning of individual rules is not clear in decision lists because the rule assumes implicitly that the bodies of the rules that precede in the lists are not satisfied. In unordered sets of rules, every rule is induced independently and its meaning is independent of other rules.

All of the above algorithms learn hypotheses for classification when training instances are given and classify all of new instances with the hypotheses. However, if the purpose of learning is only to classify new instances, it is sufficient to learn when an instance to be classified is given. Lazy learning[2] is based on this approach and learns a hypothesis for each test instance to be classified. Then, a learned hypothesis is used to classify only one test instance and a different hypothesis is learned for classifying a different test instance. LazyDT[26] is a lazy learning algorithm that learns one pseudo-optimum rule for each test instance. When large number of training instances are given, IGR induces similar rules with LazyDT because there exists a quite similar training instance to each test instance and the best rule for the training instance is the same one as the best rule for the test instance with high probability. However, IGR has the following advantages to achieve higher classification accuracy than LazyDT. First, IGR can use the class label of the target training instance to induce the rule. However, LazyDT, of course, can't use the class label of the target test instance. This makes it difficult to learn good rules by LazyDT near boundaries of classes. In addition, IGR uses plural rules to classify one instance but LazyDT induces only one rule.

5.2 Evaluation of Rules

An important problem in knowledge discovery is to evaluate rules to filter out less interesting rules because huge number of rules are embedded in typical databases. What kind of rules are interesting strongly depends on what a user of a discovery system already knows and what he/she wants to do with the discovered rules. If the user has no background knowledge of the domain of the target database, then the user prefers typical rules that summarize the characteristics of each class. However, domain experts usually know such rules and may prefer exceptional rules that represent anomalies in the given databases.

Piatetsky-Shapiro discussed the properties of rule interest criteria and proposed the following principles that the criteria should satisfy[49].

- The interestingness becomes 0 if accuracy of a rule to be evaluated coincides with the prior probability of its conclusion.
- The interestingness monotonically increases with accuracy of the rule when other parameters such as support and the prior probability of the conclusion remain the same.
- The interestingness monotonically increases with support when other parameters remain the same.
- The interestingness monotonically decreases with the probability of the conclusion when other parameters remain the same.

Roughly speaking, the above principles lead the interestingness of rules that increases with the strength of the relationship between a body and a conclusion of a rule to be evaluated. Because of the last two principles, the criteria based on these principles

prefer typical rules than exceptional rules. When evaluating a classification rule whose conclusion is a certain class label, the most interesting rule for the class with respect to the principles is the one that covers all instances of the class and rejects all instances in other classes if such a rule exists.

Many criteria of rules satisfy all of or many of these principles. For example, Smyth and Goodman proposed a discovery algorithm ITRULE[60] that uses J-measure as the criterion of rules and this criterion satisfies all of the above principles. Several discovery algorithms[3, 27, 44] extract rules that minimize the weighted sum of entropy for training instances covered by and not covered by the rule. The weighted sum of entropy satisfies the second, third and the last principles. It violates the first principle unless the prior probability of the conclusion is 50%. CN2[14] uses Laplacian error estimate to evaluate rules. This criterion also satisfies the second, third and the last principles and its generalization, m-error estimation[13] satisfies all of the four principles. Other criteria such as chi-square estimation[61], gini index[10] also satisfy the principles. These criteria of rules are the functions of support, accuracy and the prior probability of each class. Because the prior probabilities of the classes are constant, optimization of these criteria can be mapped to maximization of support and accuracy, and the difference between the criteria is how to balance support and accuracy[5]. Then, these criteria satisfy the second and the third principles. Kamber and Shinghal[33] discussed characteristic rules to represent necessity conditions and discriminant rules to represent sufficiency, and proposed the interestingness of characteristic rules based on the above principles.

As described above, the discovery algorithms based on the principles tend to extract typical rules and sometimes ignore exceptional rules. In contrast, the interestingness in Chapter 3 doesn't satisfy the third and the last principles. It doesn't depend on support

of rules and gives high scores to the exceptional rules with small support. However, the interestingness also gives high scores to typical rules with large support if the conditions of the rules are really required to achieve high accuracy. The interestingness in this thesis can be used to extract both of typical and exceptional rules.

5.3 Discovery of Exception Rules

Most knowledge discovery systems extract typical rules with large support but there are several previous works to extract interesting rules with small support. Suzuki et al.[64, 62, 63] have discussed criteria of exception rules and proposed algorithms to discover them. The most important feature of their works is to extract pairs of general and exception rules. In their first work[64], they evaluated a pair of rules by using entropy based measure, ACEP(Average Compressed Entropy Product) and proposed a discovery algorithm to extract given number of the best rule pairs with respect to ACEP. In their recent works[62, 63], they restricted the pairs of rules based on accuracy of the rules in the pairs.

Formally, a rule

$$a_1 \wedge \cdots \wedge a_L \wedge b_1 \wedge \cdots \wedge b_M \Rightarrow class = c$$

is an exception rule of its generalization

$$a_1 \wedge \cdots \wedge a_L \Rightarrow class = c'$$

if and only if

- $c \neq c'$,
- $P(class = c | a_1 \wedge \cdots \wedge a_L \wedge b_1 \wedge \cdots \wedge b_M) \geq LB_1$,
- $P(class = c' | a_1 \wedge \cdots \wedge a_L) \geq LB_2$,

- $P(class = c|b_1 \wedge \cdots \wedge b_M) \leq LB_3$,
- $P(class = c \wedge a_1 \wedge \cdots \wedge a_L \wedge b_1 \wedge \cdots \wedge b_M) \geq LB_4$,
- $P(class = c' \wedge a_1 \wedge \cdots \wedge a_L) \geq LB_5$.

The first three conditions mean that the exception rule specialized by $b_1 \wedge \cdots \wedge b_M$ implies a certain class c with high accuracy even if the general rule implies a different class c' with high accuracy. The fourth condition rejects the possibility that the conjunction $b_1 \wedge \cdots \wedge b_M$ strongly relates to the conclusion c and the exception rule is a trivial conclusion of the rule $b_1 \wedge \cdots \wedge b_M \Rightarrow c$. The last two conditions give minimum support for exception and general rules.

The interestingness I and IG proposed in Chapter 3 can be used to detect a kind of exception rules. The interestingness becomes more than one only when the rule to be evaluated concludes a certain class with high accuracy but generalizations of its body negatively relates to the class. Precisely speaking, the interestingness of a rule

$$x_1 \in D_1 \wedge \cdots \wedge x_L \in D_L \Rightarrow class = c$$

becomes high only when

$$P(c|x_1 \in D_1 \wedge \cdots \wedge x_L \in D_L) \simeq 1$$

and

$$P(c|x_1 \in D_1 \wedge \cdots \wedge x_{k-1} \in D_{k-1} \wedge x_{k+1} \in D_{k+1} \wedge \cdots \wedge x_L \in D_L) \ll P(c),$$

and these two conditions correspond to the second and the third conditions of the exception rule. In addition, large interestingness requires the expectation of accuracy

$$\frac{P(c|x_k \in D_k)P(c|x_1 \in D_1 \wedge \cdots \wedge x_{k-1} \in D_{k-1} \wedge x_{k+1} \in D_{k+1} \wedge \cdots \wedge x_L \in D_L)}{P(c)}$$

must be small and this leads the condition that $P(c|x_k \in D_k)$ is not so high. This condition corresponds to the fourth condition of exception rules. As a result, if the interestingness of

$$x_1 \in D_1 \wedge \cdots \wedge x_L \in D_L \Rightarrow class = c$$

is sufficiently high, then the rule is an exception rule of a general rule

$$x_1 \in D_1 \wedge \cdots \wedge x_{k-1} \in D_{k-1} \wedge x_{k+1} \in D_{k+1} \wedge \cdots \wedge x_L \in D_L \Rightarrow \neg c.$$

The difference between the interesting rule in this thesis and the exception rule is that the definition of the exception rule compares the rule with only one general rule and doesn't concern other generalizations of the rule. Oppositely, the interestingness defined in Chapter 3 compares with plural general rules to judge whether all conditions are really required to imply the conclusion. Because of this difference, the interestingness in this thesis sometimes rejects exception rules based on the above conditions as trivial rules. Informally, a rule $a_1 \wedge a_2 \wedge b \Rightarrow c$ is an exception rule of a general rule $a_1 \wedge a_2 \Rightarrow c'$ if accuracy of them are sufficiently high and the condition b for specializing the general rule to the exception rule doesn't have positive relation to c. However, the above definition doesn't concern the rules $a_1 \wedge b \Rightarrow c$ and $a_2 \wedge b \Rightarrow c$ that may imply the class c with high accuracy with smaller number of conditions. Suzuki ignored this possibility because the purpose of his work was to discover rule pairs. An exception rule represents an exceptional region of class c in the region of class c' corresponds to a general rule even if the exceptional region is not isolated and is a part of wider region corresponding to other general rules such as $a_1 \wedge b \Rightarrow c$ and $a_2 \wedge b \Rightarrow c$. In contrast, this thesis tries to evaluate a rule itself that corresponds to an isolated interesting area of a certain class.

Liu et al. defined direction setting (DS) rules [37] to prune useless rules. DS rules require that the conditions in their bodies negatively relate to the conclusions of the

rules. Then, DS rules represent a kind of exception rules and are highly estimated by the interestingness in this thesis. However, the definition of DS rules is qualitative but the interestingness in this thesis can evaluate rules quantitatively.

5.4 Rule Discovery in Numeric Domains

Many practical databases are represented by both of nominal and numeric attributes and discovery systems are required to extract rules from numeric attributes. For rule discovery from numeric attributes, there are two points to characterize discovery algorithms. First one is what kind of conditions are used to represent bodies of rules. This thesis focuses on the rules in which each condition only restricts values of one attribute and doesn't apply multivariate conditions that constrain combinations of values of plural attributes. The conditions on single attributes are most widely used in many learning algorithms such as C4.5[51]. However, these conditions only divide an instance space by axis-parallel hyper-plains and the learned rules can only represent hyper rectangles consist of axis-parallel hyper-plains. This restriction tends to lead complex rules when the numeric attributes are strongly correlated with each other and clusters of positive instances are oblique to the axes.

There are some, but not many researches to try this problem by using conditions on plural attributes. For example, Heath et al.[30] and Ittner and Schlosser[32] proposed decision tree learners with multivariate conditions that divide instances with linear or non-linear functions of plural attributes. Fukuda et al.[27] discussed multivariate conditions that represent x-monotone regions and rectilinear convex regions in two dimensional space defined by two numeric attributes. This thesis doesn't use these multivariate conditions because of the following three reasons. First, the multivariate conditions makes it difficult to understand what the discovered rules mean. If the

purpose of rule discovery is to support human experts to analyze the given databases, readability of the rules is quite important and the conditions with single attributes are superior than multivariate conditions. Oppositely, we don't need to concern this point and may prefer the multivariate conditions if the purpose of discovery is to classify new instances correctly and human experts don't edit the discovered rules. Second, it is difficult to combine the multivariate conditions for numeric attributes with nominal attributes. In many real world problems, discovery algorithms should extract rules with both nominal and numeric conditions and they should deal with nominal and numeric conditions within the same framework. If the discovery algorithms permit multivariate conditions on numeric attributes, then the algorithms should permit not only multivariate conditions on nominal attributes but also multivariate conditions that combine nominal and numeric attributes. Finally, multivariate conditions increase the number of possible rules significantly and makes it difficult to enumerate interesting rules.

Another issue of rule discovery in numeric domains is how to deal with possible values of numeric attributes, i.e. whether the discovery algorithms require discretization as a pre-process of rule discovery, or they directly extract rules from original non-discretized values. For classification of new instances, many algorithms learn hypotheses from original, non-discretized values. For example, C4.5 evaluates all of possible cut points for original values for every numeric attribute and selects the best condition to divide training instances at each decision node. Fayyad and Irani [23] discussed dividing training instances with plural cut points at each decision node based on minimum description principle(MDL). Instead, most discovery algorithms apply discretization to enumerate rules with given constraints on their support and accuracy. This is because enumeration with non-discretized values not only increases the

number of possible rules but also leads many quite similar rules that share the same attributes with slight different thresholds in their bodies. Such similar rules usually correspond to the same regularity in the given instances and it is sufficient to extract one or a few of them for analyzing given instances. Oppositely, discretization reduces the number of possible rules and accelerates discovery algorithms, but the discovery algorithms may miss interesting rules if domains of attributes are discretized with inappropriate thresholds. In this thesis, IGR assumes discretization and the quality of discovered rules depend on the discretization method. IIS also requires discretization but it assumes discretization into many values and the effect of discretization is not so large. All experiments in this thesis used the most simple method that discretizes a domain of each numeric attribute into values with same number of training instances but there are more sophisticated discretization algorithms. For example, chi-merge[34] starts from all distinct values appeared in training instances and continues to merge two adjacent values (intervals) that minimizes chi-squire score. Wang et al.[66] applied discretization on association rule mining and proposed discretization based on how merging two adjacent values affects interestingness of discovered association rules evaluated by J-measure. It may be possible to improve the performance of discovered rules by IGR with these discretization algorithms.

Chapter 6

Conclusions

This chapter summarizes the contributions of this thesis and discusses remaining problems for future research.

6.1 Contributions

This thesis investigated discovery of classification rules from databases of pre-classified training instances. The major purposes of rule discovery are classifying new instances correctly and analyzing characteristics of the given databases. This thesis dealt with rule discovery for both purposes.

Chapter 2 discussed discovery of rules for classifying new instances. The contribution of this chapter is a new discovery algorithm IGR(Instance Guided Rule induction) to achieve high classification ability without increasing the number of discovered rules. When discovering rules for classifying new instances, goodness of the discovered rules is basically measured by classification performance for the new instances. In addition, simplicity of the rules is also an important criterion for rule discovery. These two criteria sometimes conflict with each other. IGR extracts one rule for each training instance that works well near the training instance and combines such rules to classify new instances correctly. Empirical results showed that IGR learned more rules than standard learning algorithms such as AQ and C4.5 but was much better than the al-

gorithms with respect to classification accuracy. Bagging with C4.5 could learn rules with comparable classification accuracy with IGR but it required much more rules than IGR and was significantly slower than IGR.

Chapter 3 and 4 discussed discovery of rules for analyzing databases. The contributions of these chapters are new criteria of rules to filter out less interesting rules and two efficient discovery algorithms to extract interesting rules based on the proposed criteria in nominal and numeric domains respectively. Chapter 3 first discussed why most commonly used criteria of rules, support and accuracy, aren't sufficient to reject useless rules and presented the interestingness that evaluates a rule by comparing the real accuracy of the rule with its expectation calculated from accuracy of more general and simpler rules. Chapter 3 also presented a new discovery algorithm DIG(Discovery of Interesting rules with Grouping attribute values) to discover interesting rules efficiently from a set of training instances represented by values of nominal attributes. Chapter 4 discussed rule discovery from training instances represented by numeric attributes and presented a new discovery algorithm IIS(Iterative Interval Specialization) that searches for rules by specializing boundaries of allowed intervals for values of attributes. Chapter 3 and 4 evaluated the proposed algorithms empirically and showed that the algorithms could discover interesting rules with small number of conditions in their bodies in practical time.

6.2 Future Works

This thesis have presented the discovery algorithms and showed their usefulness. This section suggests future researches to improve them.

First, this thesis presented two discovery algorithms for analyzing given databases, DIG for nominal databases and IIS for numeric databases. However, many practical

databases are represented by both types of attributes and the discovery algorithms are required to extract rules with conditions on both of nominal and numeric attributes. DIG and IIS are quite different from each other but it is very important to unify them to analyze databases with both of nominal and numeric attributes.

Second is the use of domain knowledge. In usual, human experts have domain knowledge on the given databases and the use of the knowledge can improve the quality of discovered rules[48]. In addition, human experts sometimes expect discovery systems to justify their knowledge and to point out the lack or wrong points of their knowledge. However, all of discovery algorithms in this thesis extract rules from only training instances and can't use background knowledge. This may restrict the applications of the algorithms in this thesis.

Third, this thesis focused on conjunctive rules whose bodies are conjunctions of conditions and require all of conditions are satisfied. However, it is required to discover more powerful rules in some practical domains[45, 73, 74]. An example of such rules are the rules with threshold functions in their bodies that only require the given number (threshold) of conditions are satisfied. This type of rules become important in many real world problems such as diagnoses of diseases in which observation of more symptoms of a certain disease leads more confident diagnosis. It is possible to extend the interestingness of rules presented in Chapter 3 to the rules with threshold functions as their bodies[71]. However, it is not easy to discover such rules efficiently because support of the rules doesn't decrease as their bodies become longer and it is difficult to restrict search space with minimum support effectively.

At the end, this thesis evaluated proposed algorithms only with benchmark problems. It is quite important to apply the algorithms on real world problems and evaluate how they work and what are insufficient to discover interesting rules.

References

[1] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. I. Verkamo. Fast discovery of association rules. In U. M. Fayyad, G. P. Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, chapter 12. AAAI Press/MIT Press, 1996.

[2] D. W. Aha, editor. *Lazy Learning.* Kluwer Academic Publishers, 1997.

[3] H. Arimura, A. Wataki, R. Fujino, and S. Arikawa. A fast algorithm for discovering optimal string patterns in large text databases. In *Proceedings of the 9th International Workshop on Algorithmic Learning Theory*, Lecture Notes in Artificial Intelligence 1501, pages 247–261, 1998.

[4] T. L. Bailey and C. Elkan. Estimating the accuracy of learned concepts. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pages 895–900, 1993.

[5] R. J. Bayardo Jr. and R. Agrawal. Mining the most interesting rules. In *Proceedings of the 5th International Conference on Knowledge Discovery and Data Mining*, pages 145–153, 1999.

[6] C. L. Blake and C. J. Merz. UCI repository of machine learning databases, 1998.

[7] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth. Occam's razor. *Information Processing Letters*, 24(6):377–380, 1987.

[8] R. J. Brachman and T. Anand. The process of knowledge discovery in databases. In U. M. Fayyad, G. P. Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, chapter 2. AAAI Press/MIT Press, 1996.

[9] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[10] L. Breiman, J. Friedman, R. Olshen, and C. J. Stone. *Classification and Regression Trees.* Wadsworth Int. Group, 1984.

[11] J. G. Carbonell, R. S. Michalski, and T. M. Mitchell. An overview of machine learning. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, chapter 1. Tioga Publishing Company, 1983.

[12] R. Caruana and D. Freitag. Greedy attribute selection. In *Proceedings of the 11th International Conference on Machine Learning*, pages 28–36, 1994.

[13] B. Cestnik and I. Bratko. On estimating probabilities in tree pruning. In *Proceedings of the 6th European Working Session on Learning*, pages 138–150, 1991.

[14] P. Clark and R. Boswell. Rule induction with CN2: Some recent improvements. In *Proceedings of the 6th European Working Session on Learning*, pages 151–163, 1991.

[15] P. Clark and T. Niblett. The CN2 induction algorithm. *Machine Learning*, 3(4):261–283, 1989.

[16] W. W. Cohen. Efficient pruning methods for separate-and-conquer rule learning systems. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pages 988–994, 1993.

[17] W. W. Cohen. Fast effective rule induction. In *Proceedings of the 12th International Conference on Machine Learning*, pages 115–123, 1995.

[18] S. L. Crawford. Extensions to the CART algorithm. *International Journal of Man Machine Studies*, 1989:197–217, 1989.

[19] B. V. Dasarathy, editor. *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques.* IEEE Press, 1991.

[20] P. Domingos. Unifying instance-based and rule-based induction. *Machine Learning*, 24(2):141–168, 1996.

[21] P. Domingos. Occam's two razors: The sharp and the blunt. In *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining*, pages 37–43, 1998.

[22] U. M. Fayyad. Branching on attribute values in decision tree generation. In *Proceedings of the 12th National Conference on Artificial Intelligence*, pages 601–606, 1994.

[23] U. M. Fayyad and K. B. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pages 1,022–1,027, 1993.

[24] U. M. Fayyad, G. Piatetsky-Schapiro, and P. Smyth. From data mining to knowledge discovery. In U. M. Fayyad, G. P. Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, chapter 1. AAAI Press/MIT Press, 1996.

[25] Y. Freund and R. E. Schapire. Experiments with a new boosting algorithm. In *Proceedings of the 13th International Conference on Machine Learning*, pages 148–156, 1996.

[26] J. H. Friedman, R. Kohavi, and Y. Yun. Lazy decision trees. In *Proceedings of the 13th National Conference on Artificial Intelligence*, pages 717–724, 1996.

[27] T. Fukuda, Y. Morimoto, S. Morishida, and T. Tokuyama. Constructing efficient decision trees by using optimized numeric association rules. In *Proceedings of the 22th International Conference on Very Large Databases*, pages 146–155, 1996.

[28] J. Fürnkranz and G. Widmer. Incremental reduced error pruning. In *Proceedings of the 11th International Conference on Machine Learning*, pages 70–77, 1994.

[29] C. P. Hartmann, P. K. Varshney, K. G. Mehrotra, and C. L. Gerberich. Application of information theory to the construction of efficient decision trees. *IEEE Transactions on Information Theory*, 28(4):565–577, 1982.

[30] D. Heath, S. Kasif, and S. Salzberg. Induction of oblique decision trees. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pages 1,002–1,007, 1993.

[31] J. Hertz, A. Krogh, and R. Palmer. *Introduction to the Theory of Neural Computation.* Addison Wesley, 1991.

[32] A. Ittner and M. Schelosser. Non-linear decision trees - ndt. In *Proceedings of the 13th International Conference on Machine Learning*, pages 252–257, 1996.

[33] M. Kamber and R. Shinghal. Evaluating the interestingness of characteristic rules. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*, pages 263–266, 1996.

[34] R. Kerber. ChiMerge: Discretization of numeric attributes. In *Proceedings of the 9th National Conference on Artificial Intelligence*, pages 123–128, 1992.

[35] R. Kohavi and D. Sommerfield. Feature subset selection using the wrapper method: Overfitting and dynamic search space topology. In *Proceedings of the 1st International Conference on Knowledge Discovery and Data Mining*, pages 192–197, 1995.

[36] M. Kubat, I. Bratko, and R. S. Michalski. A review of machine learning methods. In R. S. Michalski, I. Bratko, and M. Kubat, editors, *Machine Learning and Data Mining: Methods and Application*, chapter 1. John Wiley & Sons, 1998.

[37] B. Liu, W. Hsu, and Y. Ma. Pruning and summarizing the discovered associations. In *Proceedings of the 5th International Conference on Knowledge Discovery and Data Mining*, pages 125–134, 1999.

[38] R. S. Michalski. A theory and methodology of inductive learning. In R. S. Michalski, editor, *Machine Learning: An Artificial Intelligence Approach*, chapter 4. Tioga Publishing Company, 1983.

[39] R. S. Michalski and K. A. Kaufman. Data mining and knowledge discovery: A review of issues and a multistrategy approach. In R. S. Michalski, I. Bratko, and M. Kubat, editors, *Machine Learning and Data Mining: Methods and Application*, chapter 2. John Wiley & Sons, 1998.

[40] R. S. Michalski, I. Mozetic, J. Hong, and N. Lavrac. The multi-purpose incremental learning system AQ15. In *Proceedings of the 5th National Conference on Artificial Intelligence*, pages 1,041–1,045, 1986.

[41] D. Michie, D. J. Spiegelhalter, and C. C. Taylor. *Machine Learning, Neural and Statistical Classification.* Ellis Horwood, 1994.

[42] J. Mingers. An empirical comparison of pruning methods for decision tree induction. *Machine Learning*, 4(2):227–243, 1989.

[43] T. Mitchell. Generalization as search. *Artificial Intelligence*, 18:203–226, 1982.

[44] Y. Morimoto, T. Fukuda, H. Matsuzawa, K. Yoda, and T. Tokuyama. Algorithms for mining association rules for binary segmentations of huge categorical databases. In *Proceedings of the 24th International Conference on Very Large Databases*, pages 380–391, 1998.

[45] P. M. Murphy and M. J. Pazzani. ID2-of-3: Constructive induction of m-of-n concepts for discriminators in decision trees. In *Proceedings of the 8th International Workshop on Machine Learning*, pages 183–192, 1991.

[46] S. Murthy and S. Salzberg. Lookahead and pathology in decision tree induction. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pages 1,025–1,031, 1995.

[47] S. Okamoto and N. Yugami. Generalized average-case analyses of the nearest neighbor algorithm. In *Proceedings of the 17th International Conference on Machine Learning*, pages 695–702, 2000.

[48] M. Pazzani and D. Kibler. The utility of knowledge in inductive learning. *Machine Learning*, 9:57–94, 1992.

[49] G. Piatetsky-Shapiro. Discovery, analysis, and presentation of strong rules. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, chapter 13. AAAI Press, 1991.

[50] J. R. Quinlan. Simplifying decision trees. *International Journal of Man Machine Studies*, 27:221–234, 1987.

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

[52] J. R. Quinlan. Oversearching and layered search in empirical learning. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pages 1,019–1,024, 1995.

[53] J. R. Quinlan. Bagging, boosting, and C4.5. In *Proceedings of the 13th National Conference on Artificial Intelligence*, pages 725–730, 1996.

[54] J. R. Quinlan and R. L. Rivest. Inferring decision trees using the minimum description length principle. *Information and Computation*, 80(11):227–248, 1989.

[55] J. Rissanen. Modeling by shortest data description. *Automatica*, 14:465–471, 1978.

[56] R. L. Rivest. Learning decision lists. *Machine Learning*, 2(3):229–246, 1987.

[57] L. Saitta and F. Neri. Learning in the "Real World". *Machine Learning*, 30(2):133–163, 1998.

[58] R. E. Schapire. A brief introduction to boosting. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, pages 1,401–1,406, 1999.

[59] R. E. Schapire. Theoretical views of boosting. In *Proceedings of the 4th European Conference on Computational Learning Theory*, pages 1–10, 1999.

[60] P. Smyth and R. M. Goodman. Rule induction using information theory. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, chapter 9. AAAI Press, 1991.

[61] G. W. Snedecor and W. G. Cochran. *Statistical Methods.* Iowa State University Press, 1989.

[62] E. Suzuki. Autonomous discovery of reliable exception rules. In *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining*, pages 259–262, 1997.

[63] E. Suzuki. Scheduled discovery of exception rules. In *Proceedings of the 2nd International Conference on Discovery Science*, Lecture Notes in Artificial Intelligence 1721, pages 184–195, 1999.

[64] E. Suzuki and M. Shimura. Exceptional knowledge discovery in databases based on information theory. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*, pages 275–278, 1996.

[65] E. Tsang. *Foundations of Constraint Satisfaction.* Academic Press, 1993.

[66] K. Wang, S. H. W. Tay, and B. Liu. Interestingness-based interval merger for numeric association rules. In *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining*, pages 121–127, 1998.

[67] S. M. Weiss and N. Indurkhya. *Predictive Data Mining: A Practical Guide.* Morgan Kaufmann Publishers, 1998.

[68] J. Wnek and R. S. Michalski. Hypothesis-driven constructive induction in AQ17-HCI: A method and experiments. *Machine Learning*, 14(2):139–168, 1994.

[69] N. Yugami. Theoretical analysis of Davis-Putnam procedure and propositional satisfiability. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pages 282–288, 1995.

[70] N. Yugami. Decision tree induction based on an expected accuracy. In *Proceedings of the 10th Annual Conference of JSAI, in Japanese*, pages 87–90, 1996.

[71] N. Yugami. Discovery of interesting m-of-n patterns for classification. In *Proceedings of the 3rd International Conference on Discovery Science*, pages 278–281, 2000.

[72] N. Yugami, Y. Ohta, and H. Hara. Improving repair-based constraint satisfaction methods by value propagation. In *Proceedings of the 12th National Conference on Artificial Intelligence*, pages 344–349, 1994.

[73] J. Zhang and R. S. Michalski. An integration of rule induction and exemplar-based learning for graded concepts. *Machine Learning*, 21(3):235–267, 1995.

[74] Z. Zheng. Constructing nominal X-of-N attributes. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pages 1,064–1,070, 1995.