ATTRIBUTE-ORIENTED INDUCTION

IN RELATIONAL DATABASES

by

Yandong Cai

B.Med.Sc, Jilin Medical University, 1977
M.Sc., Beijing Medical University, 1983

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

in the School
of
Computing Science

SIMON FRASER UNIVERSITY
December 1989

Approval

Name: Yandong Cai

Degree: Master of Science

Title of Thesis: Attribute-Oriented Induction in Relational Databases

Fred Popowich
Chairman

Nick Cercone
Senior Supervisor

Wo-Shun Luk
Supervisor

Bob Hadley
Supervisor

Gordon McCalla
External Examiner

December 12, 1989
Date Approved

ABSTRACT

It is beneficial as well as challenging to learn knowledge rules from relational databases because of the vast amount of knowledge implied in databases and the large amount of data stored in databases. In this thesis, we develop an attribute-oriented induction method to extract characteristic rules and classification rules from relational databases. The method adopts the artificial intelligence "learning from examples" paradigm and applies an attribute-oriented concept tree ascending technique in the learning process which integrates database operations with the learning process and provides a simple, efficient way of learning from large databases. Conjunctive rules as well as restricted forms of disjunctive rules are learned using this method. Moreover, by incorporating statistical techniques, qualitative rules with quantitative information can be learned and noisy data and exceptional cases are elegantly handled. Our analysis of the algorithms indicates that attribute-oriented induction substantially reduces the complexity of database learning processes. A prototype database learning system, DBLEARN, has been designed and implemented; early experiments with the prototype system illustrate the promise of attribute-oriented learning in relational databases.

ACKNOWLEDGMENTS

I would like to thank my senior supervisor, Professor Nick Cercone, for supervising this thesis. His work on *knowledge representation* and *expert database systems*, his teaching on *artificial intelligence*, his support, guidance and encouragement of the work, and especially his extraordinary patience while modifying and editing my thesis drafts have made this thesis a reality. I would like to thank the other members of my thesis supervisory committee: Professor Wo-Shun Luk and Professor Bob Hadley. Professor Wo-Shun Luk led me to the area of *relational database systems*. He taught me a great deal on introductory and advanced relational database topics and has given me great encouragement and support since I started my research on learning from relational databases. Professor Bob Hadley introduced me to the area of *machine learning*. He has greatly encouraged my working in this direction. He has spent a lot of his valuable time discussing many issues with me, directing me to the important problems of this research. I am indebted to Dr. Gordon McCalla for being my external examiner. Constructive comments from Dr. Gordon McCalla have helped improving the original version of the thesis. I would like to thank my husband, Professor Jiawei Han, for his valuable help on my work. His love, understanding and support have always been a source of confidence for me.

I would like to thank all the professors in the Department of Computing Sciences who have led me through the various areas of computer science in the past

few years.

I should also express my thanks to all of my friends in S.F.U. They encouraged me and helped me when I met difficulties, and made my stay in S.F.U. memorable. In particular, Howard Hamilton, discussed many interesting issues with me, and reviewed my papers for conferences. Pat T. Pattabhiraman helped me improve the presentation of the thesis and polish the English of the early version of the thesis.

I would also like to acknowledge the financial support received from Simon Fraser University and the Natural Sciences and Engineering Research Council of Canada.

Finally, I would like to thank my parents for their love, education and support since my childhood, and my sister, Yanhong Cai, for her encouragement and help.

CONTENTS

CHAPTER 1

INTRODUCTION

Learning is one of the most important characteristics of human and machine intelligence. Machine learning is a fundamental area in Artificial Intelligence which has achieved significant progress in the last two decades. Theories and algorithms for machine learning have been studied extensively [MCM83, MCM86]. Many learning systems have been constructed for scientific, business, industrial and medical applications [LBS83, MMH86, WGT87, Zyt87]. To extend machine learning to data-intensive applications, it is important to develop learning mechanisms for knowledge discovery in large databases, especially relational databases.

Relational database systems are pervasive and widely utilized in many applications [Ull89]. It is advantageous to learn characteristics of data in relational databases. By learning from databases, knowledge rules can be extracted from the large amount of data and interesting relationships among data can be discovered automatically. Moreover, relational database systems provide many attractive features for machine learning. Relational databases store a large amount of information in a structured and organized manner. Each tuple in the database can be viewed as a typed logical formula in the conjunctive normal form. Such uniformity facilitates the application of well developed database implementation techniques and the development of efficient learning algorithms in large databases.

An important machine learning paradigm, *learning from examples*, that is, learning by generalizing specific facts or observations [GeN87], has been adopted in many existing induction learning algorithms. Current systems for *learning from examples* take training examples from various sources, such as, data extracted from experiments [BuM78, Lan77, WGT87], examples given by teachers and experts [Mit77], facts recognized by people [MiC80] and rules accumulated from past experience [Qui83], etc. However, not many systems directly extract knowledge from data stored in relational databases. Since databases store a large amount of facts which can be viewed as examples for learning processes, the paradigm *learning from examples* should be the first important candidate strategy for learning from databases.

From our point of view, one of the major reasons that the current learning systems do not integrate well with relational database systems is because of the inefficiency of current learning algorithms when applying to large databases. Most existing algorithms for *learning from examples* apply a tuple-oriented approach which examines one tuple at a time. In order to discover the most specific concept that is satisfied by all the training examples, the tuple-oriented approach must test the concept coverage after each generalization on a single attribute value of a training example [DiM83, Mic83]. Since there are a large number of possible combinations in such testing, the tuple-oriented approach is quite inefficient to perform learning from large databases [Hau86]. For example, if there are 100 training examples (tuples), each tuple has 5 attributes, and each attribute value can be one of the three concepts on three different generalization levels, such coverage testing will be invoked up to $100 * 5^3 = 12500$ times in the worst case. Moreover, most existing algorithms do not

take the features and implementation techniques provided by database systems. To make learning algorithms applicable to database systems, highly efficient algorithms should be explored in depth.

In this thesis, we develop an attribute-oriented induction method for learning from relational databases. Following the *learning from examples* paradigm, our approach applies an attribute-oriented concept tree ascending technique which integrates database operations with the learning process. There are two types of knowledge rules, *characteristic rules* and *classification rules*, which can be easily learned from relational databases. The attribute-oriented induction method is demonstrated by two algorithms, an LCHR algorithm for *Learning CHaracteristic Rules* and an LCLR algorithm for *Learning CLassification Rules*. The attribute-oriented method effectively extracts both types of knowledge rules from relational databases. Our analysis of the algorithms shows that attribute-oriented induction substantially reduces the complexity of the database learning processes. Moreover, these two algorithms can learn both conjunctive rules and restricted forms of disjunctive rules, and learning can be performed with databases containing exceptions and noisy data using database statistics. Our approach provides a simple and efficient way of learning from large databases.

This thesis is organized into eight chapters. A brief survey of the methods developed for learning from examples and knowledge discovery in large databases is presented in Chapter 2. The concepts of learning from relational databases are introduced in Chapter 3; we address the primitives, the task and the characteristics in learning from databases. To demonstrate our attribute-oriented induction approach,

the LCHR algorithm and the LCLR algorithm are presented in Chapter 4 along with illustrative examples. Variations of the algorithms are discussed in Chapter 5. These variations show that our learning algorithms can be extended to cope with different learning situations. To demonstrate the power of our approach, our algorithms are analyzed and compared with other learning algorithms in Chapter 6. The implementation of the database learning algorithms and the experimental results are presented in Chapter 7. We provide concluding remarks, and propose some interesting topics for future research in Chapter 8.

CHAPTER 2

LEARNING FROM EXAMPLES: AN AI APPROACH

We survey some theoretical issues related to learning from examples, some successful models for this learning paradigm, and some recent progress in knowledge discovery in database systems and knowledge base systems which adopt the *learning from examples* philosophy.

2.1. Concepts of Learning from Examples

As a basic method in empirical learning, learning from examples has been explored extensively. We review the basic components and the generalization rules of learning from examples, the types of knowledge rules which can be learned, and the control strategies of the learning process.

2.1.1. Basic Components in Learning from Examples

Learning from examples can be characterized by a tuple <P, N, C, Λ>, where P is a set of positive examples of a concept, N is a set of negative examples of a concept, C is the conceptual bias which consists of a set of concepts to be used in defining learning rules and results, and Λ is the logical bias which captures particular logic forms [GeN87].

In most learning programs, the training examples are classified in advance by the teacher into two disjoint sets, the positive example set and the negative example set [Mic83]. The training examples represent low-level, specific information. The

learning task is to generalize these low-level concepts to general rules.

There could be numerous inductive conclusions derived from a set of training examples. For instance, the concept "red" can be generalized in several ways: "red or black", "dark color", "warm color", etc. To cope with this multiplicity of possibilities, it is necessary to use some additional information, *problem background knowledge*, to constrain the space of possible inductive conclusions and locate the most desired one(s) [GeN87]. The conceptual bias and the logical bias provide the desired concepts and logic forms which serve as this kind of background knowledge. These biases restrict the candidates to formulas with a particular vocabulary and logic forms. Only those concepts which can be written in terms of this fixed vocabulary and logic forms are considered in the learning process.

Usually, the examples presented to the learning system consist of several attributes. Depending on the structure of the attribute domains, we can distinguish among three basic types of attributes [Mic83]:

(1) Nominal attributes

The value set of such attributes consists of independent symbols or names, that is, no structure is assumed to relate the values in the domain. For example, *name, computer ID* usually do not contain structure information, and are often treated as nominal attributes.

(2) Numeric attributes

The value set of such attributes is a totally odered set. For example, *weight, salary* and *gpa* are numeric attributes.

(3) Structured attributes

The value set of such attributes has a tree structure which forms a generalization hierarchy. A parent node in such a structure represents a more general concept than the concepts represented by its children nodes. The domain of structured attributes is defined by the problem background knowledge.

For example, the attribute *shape* could be a structured attribute whose domain is a tree structure with a set of leaves:

{triangle, circle, ellipse, hexagon, square, boat, spring},

and the non-leaf nodes are defined by rules:

{circle, ellipse} $\subset$ oval

{triangle, square, hexagon} $\subset$ polygon

{oval, polygon} $\subset$ regular

{spring, boat} $\subset$ irregular

This corresponds to the concept hierarchy in Figure 2.1.

2.1.2. Generalization Rules

Learning from examples can be viewed as a reasoning process from specific instances to general concepts. The following generalization rules are particularly useful in learning systems [CoF83, Mic83].

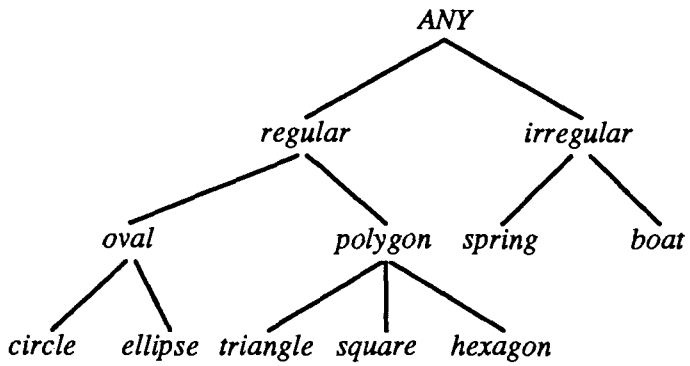


Figure 2.1. The concept tree for the structured attribute *shape*.

(1) Turning constants into variables.

If the concept F(v) holds for v when v is a constant a, or a constant b, and so on, then these concepts can be generalized into a statement that F(v) holds for every value of v. This is the rule used most often in methods of inductive inference employing predicate calculus. As a logic formula, this can be expressed as (2.1), where the notation "|<" stands for "can be generalized to".

$$F(a) \wedge F(b) \wedge \cdots \mid\!< F(v). \tag{2.1}$$

(2) Dropping conditions.

Any conjunction can be generalized by dropping one of its conjuncts. A conjunctive condition can be viewed as a constraint on the set of possible instances that could satisfy the concept. By dropping a condition, one constraint is removed and the concept is generalized. For example, the class of "red apples" can be generalized to the class of all "apples" of any color by dropping the "red" condition. This can be wrriten as:

$$red(v) \wedge apple(v) \mid< apple(v). \tag{2.2}$$

(3) Adding options.

By adding more conditions, the concept can be generalized because more instances may satisfy this concept. An especially useful form of this rule is when the alternative is added by extending the scope of permissible values of one specific concept. For example, suppose that a concept is generalized by allowing objects to be not only *red* but also *blue*. This can be expressed as follows:

$$red(v) \mid< red(v) \vee blue(v). \tag{2.3}$$

(4) Turning conjunction into disjunction.

A concept can be generalized by replacing the conjunction operator by the disjunction operator. This process is analogous to the adding-option generalization rule. This rule can be written as follows:

$$red \wedge circle \quad \mid< red \vee circle \tag{2.4}$$

(5) Climbing a generalization tree

By ascending the generalization tree, the lower level concept is substituted for by the higher level concept. This generalization rule is applicable only to the concepts whose domain is a structured value set, (that is, concepts at different levels of generality). Formally, this rule can be expressed as:

$$\left.\begin{array}{l} L(u) \in a \\ L(v) \in b \\ \quad .. \quad \in .. \\ \quad .. \quad \in .. \\ L(z) \in i \end{array}\right\} \; |\!< \; \forall(x) L(x) \in s \qquad (2.5)$$

where L is a structured attribute; a, b, ... and i are the values of u, v, ... and z in the attribute L, respectively; and s represents the lowest parent node whose descendants include nodes a, b, ... and i.

2.1.3. Types of Knowledge Rules

Given a learning-from-examples problem characterized as <P, N, C, Λ>, several different rules can be extracted. The learned concept is a *characteristic rule* if and only if it is satisfied by all of the positive examples. The learned concept is a *discriminant rule* if and only if it is not satisfied by any of the negative examples. The learned concept is an *admissible rule* if and only if it is both characteristic and discriminant [DiM83, GeN87].

For example, suppose we are given:

positive examples — a small circle, a small ellipse.

negative examples — a large ellipse, a small triangle.

We may derive several possible rules from these examples. The derived rule, "a small object", is characteristic but not discriminant, since it covers all of the positive examples but also some of the negative examples. The rule, "a circle object", is discriminant but not characteristic, since it excludes all of the negative examples but also some of the positive examples. Given the background knowledge that the higher

level concept for circle and ellipse is oval, the rule, "a small oval object", is both characteristic and discriminant and, therefore, is admissible.

Most learning algorithms are designed for learning admissible rules [DiM83, Mic83]. A few algorithms, such as INDUCE 1.2 [DiM81], SPROUTER [HaM77, HaM78], are designed for learning characteristic rules.

2.1.4. Control Strategies in Learning from Examples

Induction methods can be divided into data-driven (bottom-up), model-driven (top-down), and mixed methods depending on the strategy employed during the search for generalized concepts [DiM83]. All of these methods maintain a set, H, of the currently most plausible rules. These methods differ primarily in how they refine the set H so that it eventually includes the desired concepts.

In the data-driven methods, the presentation of the training examples drives the search. These methods process the input examples one at a time, gradually generalizing the current set of concepts until a final conjunctive generalization is computed. The typical examples of such control strategy include the candidate-elimination algorithm [Mit77, Mit79], the approach adopted in [HaM77, HaM78, Ver75, Win75], the ID3 technique of Quinlan [Qui86], and the Bacon learning system [Lan77].

In the model-driven methods, an a priori model is used to constrain the search. These methods search a set of possible generalizations in an attempt to find a few "best" hypotheses that satisfy certain requirements. Typical examples of systems which adopt this strategy are AM [Len77], DENDRAL and Meta-DENDRAL [BuM78], and the approach used in the INDUCE system [DiM81].

Data-driven techniques generally have the advantage of supporting incremental learning. The learning process can start not only from the specific training examples, but also from the rules which have been discovered. The learning systems are capable of updating the existing hypotheses to account for each new example. In contrast, the model-driven methods, which test and reject hypotheses based on an examination of the whole body of data, are difficult to be used in incremental learning situations. When new training examples become available, model-driven methods must either backtrack or restart the learning process from the very beginning, because the criteria by which hypotheses were originally tested (or schemas instantiated) have been changed [DiM83].

An advantage of model-driven methods, on the other hand, is that they tend to have good noise immunity. When a set of hypotheses, *H*, is tested against noisy training examples, the model-driven methods need not reject a hypothesis on the basis of one or two counterexamples. Since the whole set of training examples is available, the program can use statistic measures of how well a proposed hypothesis accounts for the data. In the data-driven method, the set of hypotheses, *H*, is revised each time on the basis of the current training example. Consequently, a single erroneous example can cause a large perturbation in *H* (from which it may never recover) [DiM83].

2.2. Some Successful Models in Learning from Examples

Since the 1960's, many algorithms and experimental systems on *learning from examples* have been developed [MCM83, MCM86], which demonstrated aspects of machine learning in science, industry and business applications

[Hau87a, Ren86, WaE87, WGT87]. In this section, we present several successful models which are related to our research work.

2.2.1. Candidate Elimination Algorithm

Mitchell developed an elegant framework, *version space*, for describing systems that use a data-driven approach to concept learning [Mit82]. This framework can be described as follows. Assume we are trying to learn some unknown target concept defined on the instance space. We are given a sequence of positive and negative examples which are called samples of the target concept. The task is to produce a concept that is consistent with the samples. The set of all hypotheses, H, that are consistent with the samples is called the version space of the samples. The version space is empty in the case that no hypothesis is consistent with the samples.

Mitchell proposed an algorithm, called candidate-elimination algorithm, to solve this learning task. The algorithm maintains two subsets of the version space: the set S of the most specific hypotheses in the version space and the set G of the most general hypotheses. These sets are updated with each new example. The positive examples force the program to generalize the S set, and the negative examples force the program to specify the G set. The learning process terminates when $G = S$.

A good feature of this method is that the incremental learning can be performed by the learning program. The sets S and G can easily be modified to account for new training examples without any recomputation.

However, as with all data-driven algorithms, the candidate elimination algorithm has difficulty with noisy training examples. Since this algorithm seeks to find a

concept that is consistent with all of the training examples, any single bad example (that is, a false positive or false negative example) can have a profound effect. When the learning system is given a false positive example, for instance, the concept set becomes overly generalized. Similarly, a false example causes the concept set to become overly specialized. Eventually, noisy training examples can lead to a situation in which there are no concepts that are consistent with all of the training examples.

The second and most important weakness of this algorithm is its inability to discover disjunctive concepts. Many concepts have a disjunctive form, but if disjunctions of arbitrary length are permitted in the representation language, the data-driven algorithm described above never generalizes. Unlimited disjunction allows the partially ordered rule space to become infinitely "branchy".

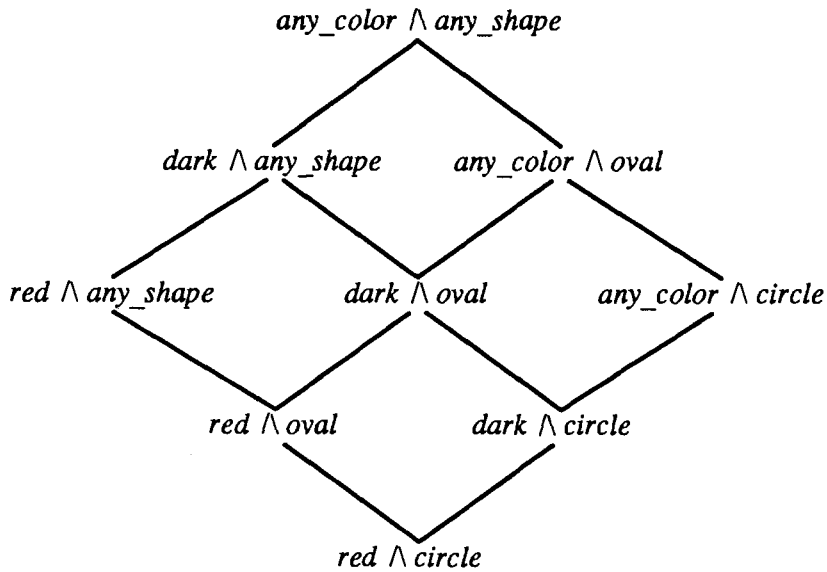
There are two computational problems associated with this method. The first one is that in order to update the sets S and G we must have an efficient procedure for testing whether or not one hypothesis is more general than another. Unfortunately, this testing problem is NP-complete if we allow arbitrarily many examples and arbitrarily many attributes in hypotheses [HaM78]. The second computation problem is that the size of the sets S and G can become unmanageably huge. It has been shown that, if the number of attributes is large, the size of set S and set G can grow exponentially in the number of examples [Hau86].

To improve computational efficiency, Haussler proposed a one-sided algorithm which is in contrast to the two-sided approach of the candidate elimination algorithm.

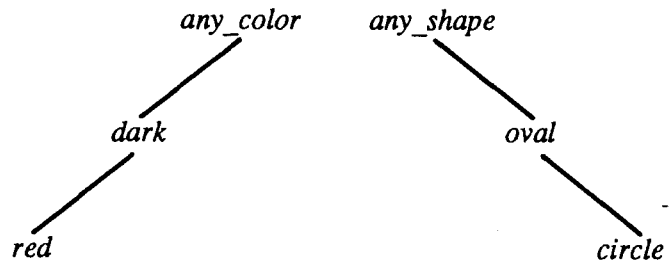
The one-sided algorithm computes only the set S using the positive examples and then checks to see if any negative examples are contained in the set S. If the rule in the set S is not satisfied by any negative examples, the rule is valid. Otherwise, there is no rule which can be discovered [Hau86, Hau87b].

In some learning situations, it is possible for the user to select training examples and to acquire information about their classification. In this case, a common strategy to maximize the learning performance is to select an example that halves the number of candidate formulas, that is, one that satisfies one-half of the candidates and does not satisfy the other half. The advantage of this strategy is that, by getting the classification of such an example, we can eliminate one-half of the remaining candidates. However, the main problem with the halving strategy is computational expense. In the worst case, we need to compare each example with each concept to determine whether or not the example satisfies the concept. If there are m examples and n candidates, then in the worst case we need mn steps to select the best example. This is infeasible when either m or n is very large.

Subramanian and Feigenbaum proposed a method, experiment generation, to solve this problem [SuF86]. They proposed to partition an instance into several independent sub-instances and to factor the entire version space into multiple separate, smaller version spaces. The test procedure for selecting the best training instance can be first performed in each factored version space, and then the resulting "sub-instance" can be combined into a single instance to be tested. The computational advantages of factoring are striking. Suppose that a version space can be fac-



a) The entire version space



b) The factored version spaces

Figure 2.2. The version spaces for the positive example "red ∧ circle".

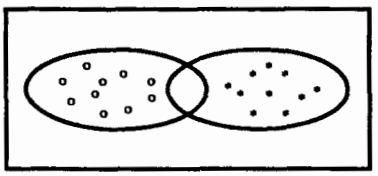
tored into k factors, with p nodes each. Whenever this is the case, the size of the unfactored version space must be p^k. If we can factor the version space, then we can "factor" each instance into k parts, one for each factor of the version space. If there are q possibilities for each part, then there must be q^k instances. The total cost for selecting a training instance without factoring is $p^k\ q^k$, whereas the total cost with factoring is just kpq, a substantial saving when p or q is large. Figure 2.2 shows the

entire version space and the factored version spaces in which the training example "red $\wedge$ circle" is the sole positive example. While the entire version space contains 9 nodes, the factored version spaces consist of only 6 nodes.

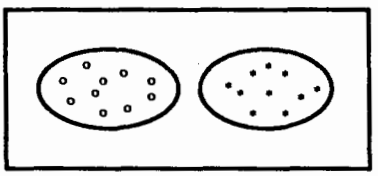
2.2.2. AQ11 and AQ15 systems

Michalski and his colleagues have developed a series of AQ learning systems. The AQ11 system [MiC80] is designed to find the most general rule in the rule space that discriminates training examples in a class from all training examples in all other classes. Michalski et. al. call these types of rules *discriminant descriptions* or *discriminant rules* since their purpose is to discriminate one class from a predetermined set of other classes.

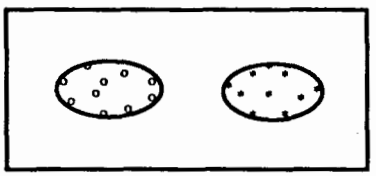
The language used by Michalski to represent discriminant rules is VL1, an extension of the propositional calculus. VL1 is a fairly rich language that includes conjunction, disjunction, and the set-membership operators. Consequently, the rule space of all possible VL1 discriminant rules is quite large. To search this rule space, AQ11 uses the AQ algorithm, which is nearly equivalent to the repeated application of the candidate-elimination algorithm. AQ11 converts the problem of learning discriminant rules into a series of single-concept learning problems. To find a rule for class A, it considers all of the known examples in class A as positive examples and all other training examples in all of the remaining classes as negative examples. The AQ algorithm is then applied to find a concept that covers all of the positive examples without covering any of the negative examples. AQ11 seeks the most general such concept, which corresponds to a necessary condition for class membership.



a) The most general rules



b) The nonoverlapping rules



c) The most specific rules

Figure 2.3. Three different types of discriminant rules.

The discriminant rules may overlap in regions of the examples that have not yet been observed, as shown in Figure 2.3a. AQ11 also has a method for finding a nonoverlapping set of classification rules, which is schematically illustrated in Figure 2.3b. The AQ algorithm has the power of performing incremental learning. This algorithm can accept not only training examples (as represented by very specific points in the

rule space) but also generalized concepts that are conjuncts in the rule space corresponding to sets of training examples. This allows AQ11 to treat the concepts themselves as negative examples when it is learning the concept for a subsequent class.

The discriminant rules developed by AQ11 correspond (roughly) to the set of most general concepts consistent with the training examples. In many situations, it is also good to develop the most specific concepts of that class (Figure 2.3c), thus permitting a very explicit handling of the unobserved portions of the space.

When the most general concept and the most specific concept are both available, *definite classification* (the examples are covered by the most specific concept), *probable classification* (the examples are covered by the most general concept), and *multiple classification* (the examples are covered by several most general concepts) can be chosen to be performed.

Michalski and his colleagues conducted an interesting experiment to compare the quality of rules for soybean disease obtained through expert consultation to rules developed by the learning process [MiC80]. Surprisingly, the computer-generated rules outperformed the expert-derived rules. Furthermore, the computer-derived rules tended to list fewer alternative diagnoses. This experimental result shows that automatic rule induction can, in some situations, lead to more reliable and more precise diagnosis rules than those obtained by consultation with the expert.

After developing the AQ11 system, Michalski et. al. proposed another inductive learning system AQ15 in 1986 [MMH86, Mic87]. This system is an extended version

of the AQ11 system, which is able to incrementally learn disjunctive concepts from noisy and overlapping examples, and can perform constructive induction in which new concepts are introduced in the formation of inductive conclusions.

To accommodate uncertainty in the learning process, AQ15 generates rules that have a pair of weights associated with them, t-weight and u-weight. The t-weight represents the *total* number of examples (events) explained by the rule, and the u-weight represents the number of examples explained *uniquely* by that expression, respectively. For example, the following rule

$$[\text{Transport} = \text{car}] <= [\text{Weather\_type} = \text{cloudy} \vee \text{rain}] \wedge [\text{Temp} = 40..60]$$

(t-weight:40, u-weight:22)

represents that there are 40 events that satisfy this rule, that is, if the weather is cloudy or raining, and the temperature is 40 to 60 degrees, the means of transportation should be a car. Among these 40 events, 22 events can only satisfy this rule, and 18 events can not only satisfy this rule, but also some other rules.

The t-weight may be interpreted as a measure of the representativeness of a concept. The u-weight may be interpreted as a measure of importance of the concept. The concepts with very low u-weights can be viewed as describing rear, exceptional cases. If the learning examples from which rules are derived are noisy, such "light" concepts may be indicative of errors in the data.

To eliminate rules which are the least reliable and which represent the weakest correlations found between attributes and classes, AQ15 performs "rule truncation" after induction. The rules which have a light t weight are cut off. A *flexible matching*

procedure is used whenever these truncated rules are applied. When matching a new example against a set of decision rules, if there are several matches or no match, the system activates the flexible evaluation schema that uses statistical techniques to determine the best match (or the most probable one).

In an experimental application to three medical domains, the AQ15 program learned decision rules that performed at the level of accuracy of human experts. A surprising and potentially significant result is the demonstration that the complexity of the knowledge base can be drastically decreased without affecting its performance accuracy.

2.3. Knowledge Discovery in Large Databases and Knowledges-Base

Currently, the steady growth in the number and size of large databases in many areas, including medicine, business and industry has created both a need and an opportunity for extracting knowledge from databases. Some recent results have been reported which extract different kinds of knowledge from databases.

Knowledge discovery in databases poses challenging problems, especially when databases are large. Such databases are usually accompanied by substantial domain knowledge to facilitate discovery. Access to large databases is expensive - hence it is necessary to apply the techniques for sampling and other statistical methods. Furthermore, knowledge discovery in databases can benefit from many available tools and techniques in different fields, such as, expert systems, machine learning, intelligent databases, knowledge acquisition, and statistics [Lub89, Pia89].

2.3.1. INLEN System

The INLEN system was developed by Kaufman et. al. in 1989 [KMK89]. The name INLEN derives from **In**ference and **Learn**ing. This system combines database, knowledge-base, and machine learning techniques to provide a user with an integrated system of tools for conceptually analyzing data and searching for interesting relationships and regularities among data. It merges several existing learning systems and provides a control system to facilitate access. Figure 2.4 illustrates the organization of the system.

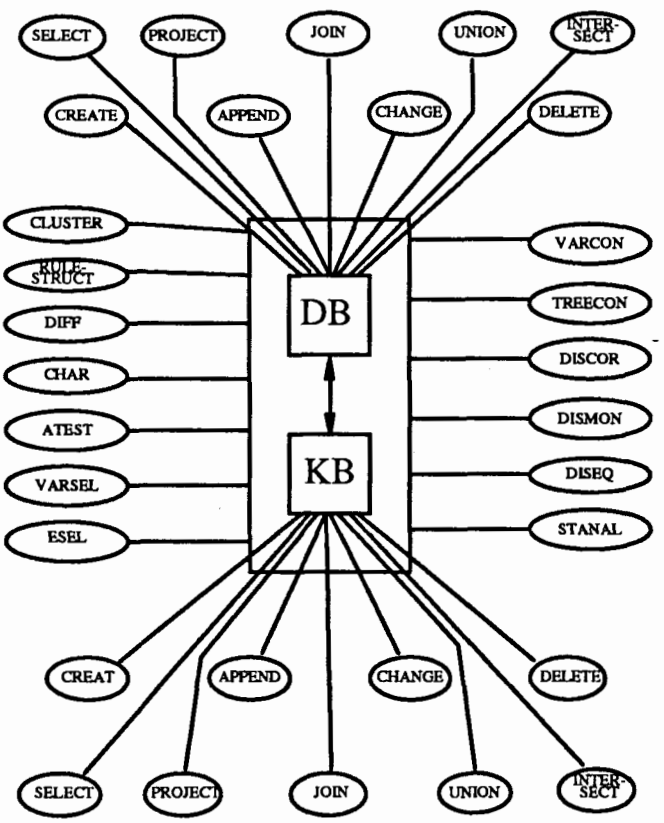


Figure 2.4. The organization of the INLEN System.

The INLEN system consists of a relational database for storing known facts about a domain, and a knowledge base for storing rules, constraints, hierarchies, decision trees, equations accompanied with preconditions, and enabling conditions for performing various actions on the database and/or knowledge base. The knowledge base can contain not only the knowledge about the contents of the database, but also meta-knowledge for the dynamic upkeep of the knowledge base itself.

INLEN employs three sets of operators: data management operators (DMOs), knowledge management operators (KMOs), and knowledge generation operators (KGOs). The DMOs are standard operators for accessing, retrieving and manually altering the information in the database. The KMOs are used to create, manipulate and modify INLEN's knowledge base, thereby allowing the knowledge base to be handled in a manner analogous to handling a database. The KGOs take input from both the database and knowledge base, and invoke various machine learning programs to perform learning tasks. For example, the operator CLUSTER creates conceptual classifications of data, which is based on the conceptual clustering algorithm developed in [MiS83]. The operator DIFF determines the discriminant rules, which can be executed in an AQ program [MMH86]. The operator CHAR discovers the characteristic rules, which is also implemented in an AQ program [Mic83]. The operator VARSEL selects the most relevant attributes and the operator ESEL determines the most representative examples. The operator DISEQ discovers equations governing numeric variables, which is based on the ABACUS-2 system for integrated qualitative and quantitative discovery [FaM86]. ABACUS-2 is related to programs such as BACON [LBS83], FAHRENHEIT [Zyt87] and COPER [Kok86]. Most of

these machine learning programs invoked by KGOs are existing learning algorithms which have been well implemented.

As in the case of many machine learning systems, the major challenge to the INLEN system is the computational inefficiency. Many learning algorithms included in this system adopt the tuple-oriented approach which examines the training examples tuple by tuple. In the learning process, these algorithms usually have a large search space and costly time complexity because they are not designed for large databases. Although this system integrates database, knowledge-base and machine learning techniques, the database operations are applied only for retrieving data and storing knowledge rules. The algorithms in this system do not take advantage of database implementation techniques in the learning processes.

2.3.2. An Algorithm for Discovering Strong Rules in Databases

Another interesting study on learning from relational databases was performed by Piatetsky-Shapiro [Pia89]. He develped an algorithm to discover strong rules in the relational databases. Somewhat different from an *exact* rule, which is a rule that is always correct, a *strong* rule is one that is always or almost always correct. This algorithm can find interesting strong rules of the form $(A = a) \rightarrow (B = b)$ from relational databases, that is, if the value of attribute A is a, then the value of attribute B is b.

This algorithm requires only one access to each database tuple. It is thus optimal to within a constant factor, since at least one access is needed to each tuple to check whether this tuple disproves any of the previously inferred rules.

The idea is to hash each tuple according to the value of A. When a tuple is hashed to an empty cell, the cell is initialized. Each cell contains the value of A, the *Count* of tuples hashed to that cell and a current cell *Tuple*. When a tuple is hashed to an occupied cell, it is compared with the cell *Tuple* and the comparison result is stored in the cell *Tuple*. At the end of hashing a cell for $(A = a)$ contains all the information necessary for deriving rules implied by $(A = a)$, such as, the number of tuples whose value of attribute A are a and the differences among those tuples which are hashed to the same cell.

A significant speed-up is achieved by using a test for early rejection of rules in an attribute. For a nominal attribute, if the value in the newly hashed tuple is different from the value stored in the cell *Tuple*, this attribute can be removed from further consideration. A taxonomic or an interval attribute is rejected when the intermediate result covers more than a user specified threshold value which is the maximum allowed sample coverage.

Piatetsky-Shapiro has derived formulas for predicting rule accuracy on the entire database after rules are discovered in a sample. These formulas measure the significance of the correlation between two attributes based on some statistical techniques.

This algorithm has been implemented in LISP and tested in relational databases. While most machine learning algorithms suffer from computational inefficiency, this algorithm can discover many strong rules from databases quickly, and can therefore be applied to relatively large databases. However, this algorithm may generate a large

set of rules. For example, the author conducted an experiment on 500 tuples, each having 12 attributes, and the learning algorithm produced 150 rules [Pia89]. This system cannot perform incremental learning when the database is updated. The learning process must be restarted after the new data are inserted into a database, because the criteria which determine whether a tuple should be rejected or saved have been changed.

CHAPTER 3

CONCEPTS OF LEARNING FROM DATABASES

We introduce the concept primitives for learning from databases, discuss the types of rules that can be learned from databases, and then present an example to illustrate these ideas.

3.1. Primitives of Learning from Databases

Learning from databases can be characterized by a triple <D, C, Λ>, where D represents the set of data in the database relevant to a specific learning task, C represents a set of "concept biases" (generalization hierarchy, etc.) useful for defining particular concepts, and Λ is a language used to phrase definitions.

3.1.1. Data Relevant to Learning Task

Although a relational system stores a large amount of data, usually only a portion of it is relevant to a specific learning task. For example, to characterize the features of graduate students, only the data relevant to graduate students are useful in the learning process. Similarly, to distinguish graduate students from undergraduate students, it is only necessary to consider the tuples for students. If the relevant data are spread over several relations, appropriate operations should be performed on those relations to obtain a new relation before the learning algorithm is applied. Thus, D, the set of data in the database relevant to a specific learning task, should be obtained by performing relational operations, such as selection, projection and join,

to collect the necessary data for learning. In this thesis, we assume that such preprocessing has been performed, and we focus only on one data relation relevant to our learning process.

Most *learning from examples* algorithms partition the set of examples into positive and negative sets. The positive examples are used to generalize the learning concepts, and the negative examples are used to specialize the learning concepts [DiM83, Mic83]. However, since a relational database usually does not explicitly store negative data, no specified negative examples can be used to perform the specialization process in learning. All of the data stored in the database which characterize the features of a property are positive data. However, most database applications assume that all of the information about a property is stored in the database. Therefore, negative data can be derived based on *the closed world assumption* [Rei84]. For example, to distinguish graduate students from undergraduate students, the properties which belong to undergraduate students can be viewed as negative data.

3.1.2. Conceptual Bias Useful for Defining Concepts

It is often necessary to incorporate higher level concepts in the learning process [GeN87, Rus88]. As in most learning processes, candidate rules are restricted to formulas with a particular vocabulary, that is, a basis set called the *conceptual bias*, permitting the learned rules to be represented in a simple and explicit form. Different levels of concepts can be organized into a taxonomy of concepts. The concepts in a taxonomy can be partially ordered according to general-to-specific ordering. The most general point is the null description (described by a reserved word "ANY"), and

the most specific points correspond to the specific values of attributes in the database. The specification of conceptual bias is a necessary and natural process for learning. Usually, the conceptual bias should be provided by knowledge engineers or domain-specific experts. We assume that conceptual clustering produces a taxonomic hierarchy of classes of similar objects in which the subclasses of each class are mutually exclusive and jointly exhaustive. In our discussion, the conceptual bias is given for each attribute which is represented as a concept tree. Such a concept tree is specified using an IS-A hierarchy and stored in a relation table, the *concept hierarchy table*. Other methods for automatically (or semi-automatically) obtaining a concept hierarchy in the database are discussed in Chapter 6.

3.1.3. Language Used to Phrase Definitions

We use first-order predicate calculus as the primitive language for learning from databases. From the logical point of view, each relation tuple is a formula in the conjunctive normal form. For example, the tuple

Name	Category	Major	Birth_Place	GPA
Jackson	senior	computing	Vancouver	3.5

can be viewed as representing the logic formula:

$$\exists\, t\ (\ (Name\,(t) = \text{Jackson}) \wedge\ (Category\,(t) = \text{senior}) \wedge (Major\,(t) = \text{computing})$$
$$\wedge\ (Birth\_Place\,(t) = \text{Vancouver}) \wedge (GPA\,(t) = 3.5)\ ).$$

The intermediate and final learning results can also be represented using relational tables. Such a relation is called a *generalized relation.*

Definition: A *generalized relation* is a relation obtained by substituting the specific concept(s) by the general concept(s) in some attribute(s).

The final generalized relation may contain several tuples which represent a disjunction of several conjunctions (tuples). Therefore, our logic bias [GeN87] on the learned rules (hypotheses) is not limited to conjunctive definitions but to a small number of disjunctions. Such a relaxation makes learning more effective because it is often necessary to represent the learning results in some disjunctive form. A maximum number of disjunctions of the resulting formula, that is, the maximum number of tuples in a final generalized relation, can be specified by users as a *threshold* value of the learning process. For example, if a threshold value is three, the learning process will derive a rule consisting of at most three disjunctions with each being a sequence of conjuncts.

3.2. Two Types of Rules

There are two types of rules, *characteristic rules* and *classification rules*, which can be easily learned from relational databases.

Definition: A *characteristic rule* is an assertion which characterizes the concepts satisfied by all of the data stored in the database.

For example, the symptoms of a specific disease can be summarized as a characteristic rule.

Definition: A *classification rule* is an assertion which discriminates the concepts of one class from other classes.

For example, to distinguish one disease from others, a classification rule should summarize the symptoms that discriminate this disease from others.

Both characteristic rules and classification rules are useful in many applications. A characteristic rule provides generalized concepts about a property which can help people recognize the common features of the data in a class. The classification rule gives a discriminant criterion which can be used to predict the class membership of new data.

Since learning these two rules represents two different learning tasks, different sets of examples are required for the learning processes. The characteristic rules only concern the characteristics of the data. Therefore, positive examples alone are enough to furnish the learning task. However, for learning classification rules, the negative examples must be incorporated into the learning process to derive the concepts which have the discriminant property.

The data relevant to the learning task can usually be classified into several classes based on the values of a specific attribute. For example, the data about students may be classified into graduate students and undergraduate students based on the value of the attribute "Category". We introduce new concepts *target class* and *contrasting class*.

Definition: A *target class* is a class in which the data are tuples in the database consistent with the learning concepts.

Definition: A *contrasting class* is a class in which the data do not belong to the target class.

For instance, to distinguish graduate students from undergraduate students, the class of graduate students is the target class, and the class of undergraduate students is the contrasting class.

3.3. An Example

To illustrate these ideas, Table 3.1 is given as a relation of a sample university database.

Example 3.1. "*Student*" is a relation of a sample university database with attributes Name, Category, etc. Suppose that our task is to learn characteristic rules of graduate students.

Name	Category	Major	Birth_Place	GPA
Anderson	M.A.	history	Vancouver	3.5
Bach	junior	math	Calgary	3.7
Carey	junior	liberal arts	Edmonton	2.6
Fraser	M.S.	physics	Ottawa	3.9
Gupta	Ph.D.	math	Bombay	3.3
Hart	sophomore	chemistry	Richmond	2.7
Jackson	senior	computing	Victoria	3.5
Liu	Ph.D.	biology	Shanghai	3.4
Meyer	sophomore	music	Burnaby	3.0
Monk	Ph.D.	computing	Victoria	3.8
Wang	M.S.	statistics	Nanjing	3.2
Wise	freshman	literature	Toronto	3.9

Table 3.1. A relation *Student* in a sample university database.

Clearly, only the facts related to "graduates" in the database are relevant to this learning task. Therefore, to obtain the relevant set of data, D, selection should be performed on the database. However, since there is no explicit attribute value "graduate" stored in the student "Category", a concept hierarchy table specifying the rela-

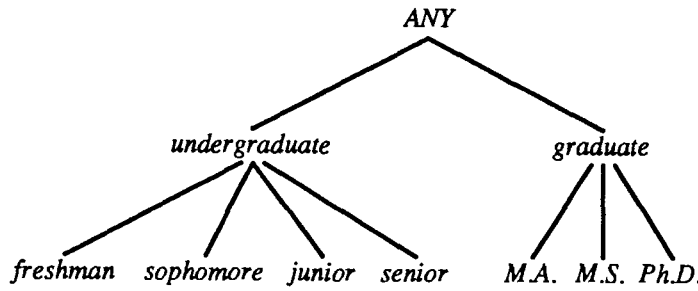
tionship between the values in category and "graduates" should be consulted in order to extract the relevant set of data.

We then examine C, the conceptual bias. Suppose the concept hierarchy table of Figure 3.1 is specified in the university database, where A $\subset$ B indicates that B is a "generalization" of A.

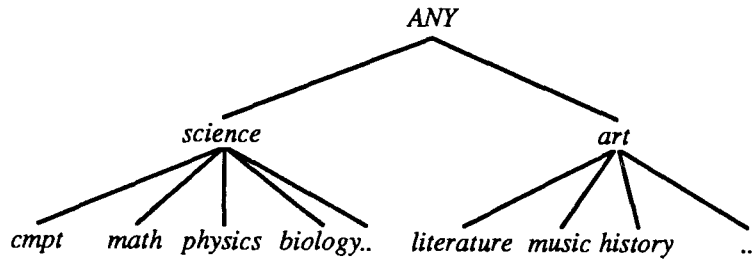
{ computing, math, biology, chemistry, statistics, physics } $\subset$ science
{ music, history, liberal arts, literature } $\subset$ art
{ freshman, sophomore, junior, senior } $\subset$ undergraduate
{ M.S., M.A., Ph.D. } $\subset$ graduate
{ Burnaby, Richmond, Vancouver, Victoria } $\subset$ British Columbia
{ Calgary, Edmonton } $\subset$ Alberta
{ Ottawa, Toronto } $\subset$ Ontario
{ Bombay } $\subset$ India
{ Shanghai, Nanjing } $\subset$ China
{ China, India } $\subset$ Foreign
{ British Columbia, Alberta, Ontario } $\subset$ Canada
{ 2.0 — 2.9 } $\subset$ average
{ 3.0 — 3.4 } $\subset$ good
{ 3.5 — 4.0 } $\subset$ excellent

Figure 3.1. A concept hierarchy table of the university database

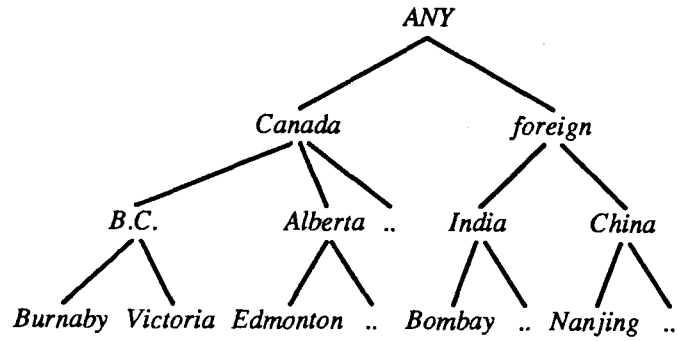
The specified concept hierarchy represents a taxonomy of concepts of the values in an attribute domain, which can be organized as a concept tree for each attribute domain. The four concept trees in relation "*Student*" are in Figure 3.2.



a) Concept tree for "Category"



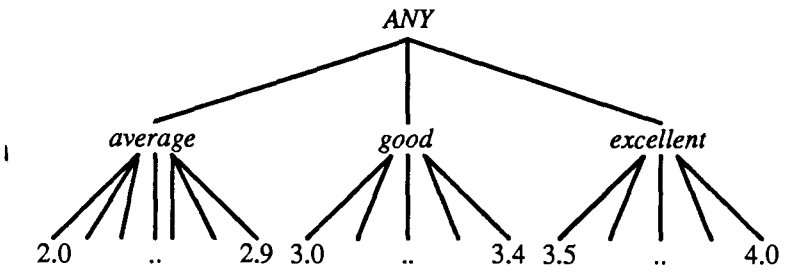
b) Concept tree for "Major"



c) Concept tree for "Birth_Place"

(to be continued)

(continued)



d) Concept tree for "GPA"

Figure 3.2. Concept trees for the four attributes

After consulting the appropriate concept tree, the set of data about "graduates" can be obtained by selection of only those students who are in the category of graduates, that is, their category values are in { M.S., M.A., Ph.D. }. Retrieved data are shown in Table 3.2.

Name	Category	Major	Birth_Place	GPA
Anderson	M.A.	history	Vancouver	3.5
Fraser	M.S.	physics	Ottawa	3.9
Gupta	Ph.D.	math	Bombay	3.3
Liu	Ph.D.	biology	Shanghai	3.4
Monk	Ph.D.	computing	Victoria	3.8
Wang	M.S.	statistics	Nanjing	3.2

Table 3.2. The set of data relevant to "graduates"

Finally, we examine the language Λ. Each positive instance is stored as a tuple in the data relation, which can be viewed as a logic formula in conjunctive normal form. For example, each tuple in Table 3.2 is a conjunction of 5 conjuncts. After the learning process, the learned rule is represented by a small number of tuples (≤ threshold) in the final generalized relation. Each tuple is in conjunctive normal form and

the relationship among these tuples is disjunction. For example, Table 3.3 shows the final generalized relation derived by the learning process from data D (Table 3.2) and the given conceptual bias (Figure 3.1 and Figure 3.2).

Major	Birth_Place	GPA
ANY	Canada	excellent
science	Foreign	good

Table 3.3. The learned rule in relation table form

This final generalized relation corresponds to a rule which is in a disjunctive normal form with 2 disjuncts, and each disjunct is a conjunction of 3 conjuncts.

CHAPTER 4

ATTRIBUTE-ORIENTED INDUCTION IN RELATIONAL DATABASES

Efficient induction techniques in relational databases are challenged by the large size of relational databases. Most existing algorithms for learning from examples conduct exhaustive searches of the given concept space, which makes the algorithms infeasibly slow for large database applications [CoF83]. Furthermore, although relational databases provide many facilities which have been well implemented, most machine learning algorithms do not take advantage of these facilities. Those learning systems suffer from computational inefficiency when they are used for learning from relational databases.

To make the learning mechanism applicable in relational databases, the learning algorithm should be able to utilize the database implementation techniques and compute efficiently. We develop an attribute-oriented induction approach which can effectively learn the characteristic rules and classification rules from relational databases [CCH89b]. Our approach integrates database operations with the learning process and provides a simple and efficient way of learning from large databases. In contrast to the tuple-oriented approach, the attribute-oriented approach performs generalization attribute by attribute. The training data are examined one attribute at a time. After the generalized sub-concepts on each attribute have been generated, the sub-concepts are combined to form the entire concept. Our approach is demonstrated by two algorithms, the LCHR algorithm and the LCLR algorithm.

4.1. Learning Characteristic Rules

Since a large number of examples stored in the database usually provides information rich enough to characterize a property, it is important to learn characteristic rules from databases.

The first algorithm we developed is the LCHR (Learning CHaracteristic Rules from relational databases) algorithm [CCH89a]. In this algorithm, an attribute-oriented concept tree ascending technique is applied which substitutes the lower-level concept of the attribute in a tuple by its corresponding higher-level concept and thus generalizes the relation. As a result, different tuples may be generalized to the same concept. By eliminating identical tuples and using a threshold value to control the generalization process, the final generalized relation consists of only a small number of tuples, which can be transformed to a simple logic formula. We examine such a generalization procedure by analyzing the learning process in Example 3.1.

Since the task of Example 3.1 is to learn the characteristic rule for graduate students, it is unimportant to distinguish M.S. students and Ph.D. students. As long as "graduate students" have been selected as the set of relevant data shown in Table 3.2, the attribute "Category" can be removed in the learning process.

The first attribute, "Name", is the key of the relation. Since each key or candidate key is distinct in a relation, it represents a large set of distinct values which should be generalized. If there is no higher level concept provided in the concept tree, the attribute should be removed in the learning process. This process can also be viewed as first generalizing the values to "ANY" (or "null" description), and then

removing the attribute since "ANY" does not provide interesting information on the attribute. Removal of an attribute may also apply to a non-key attribute under similar conditions. Therefore, we have

Generalization Strategy 4.1. (Attribute Removal) *If there is a large set of distinct values for an attribute but there is no higher level concept provided for the attribute, the attribute should be removed during generalization.*

Reasoning.

Removing attributes corresponds to the generalization rule, *dropping conditions*, in *learning from examples* [Mic83]. Consider a tuple as a set of conjuncts in the logic forms, and an attribute value as one of the conjuncts. By removing a conjunct, we eliminate a constraint and thus generalize the rule. If there is a large set of distinct values for an attribute, the large set of values must be generalized, because the learning task is to derive the generalized concepts. However, if there is no higher level concept provided for the attribute, it cannot be generalized by ascending the concept tree. Therefore, the attribute should be removed. □

The key of a relation may consist of a set of decomposable components. For example, two components, "student-id" and "course-id", form a composite key for the relation "student-course-grade". Generalization should be performed on each component of the composite key. Moreover, generalization may even be performed on a single key attribute if the key consists of decomposable subcomponents. For example, "student-id" may encode information about starting year and department, thus generalization can still be performed on such a single key to obtain concepts like

"computer science students" or "new students". To simplify subsequent discussion, we assume that any single attribute value is atomic (not decomposable). In general, we have

Generalization Strategy 4.2. (Generalization on the smallest decomposable components) *Generalization should be performed on the smallest decomposable components of a data relation.*

Reasoning.

Learning characteristic rules is essentially a process of learning from positive examples only. The *least commitment principle* (that is, commitment to minimally generalized concepts) should be enforced for effective learning. By generalizing the least decomposable components, we may discover the relationships among such components without losing information about their composite components. □

We then examine the remaining three attributes. None of these attributes is a key or a candidate key. However, each attribute contains many distinct values and is associated with some higher level concept in the concept tree of Figure 3.2. Clearly, substituting the value of an attribute by its higher level concept in the concept tree generalizes the rule, e.g., from "physics" to "science" and from "Vancouver" to "B.C.". In general, we have

Generalization Strategy 4.3. (Ascending the concept tree) *If there are many distinct values for an attribute and there exists a higher level concept in the concept tree for the attribute, each value in the attribute of the relation should be substituted by a higher level concept in the learning process.*

Reasoning.

This strategy corresponds to the generalization rule, *climbing generalization trees* [Mic83]. The substitution of an attribute value by its higher-level concept makes the tuple cover more cases than the original one and thus generalizes the tuple. Ascending the concept tree one level at a time ensures that the least commitment principle is followed and the overgeneralization is avoided. □

By removing two attributes and generalizing the three remaining ones, the relation depicted in Table 3.2 is generalized to a new relation illustrated in Table 4.1 (with redundant tuples eliminated).

Major	Birth_Place	GPA
art	B.C.	excellent
science	Ontario	excellent
science	B.C.	excellent
science	India	good
science	China	good

Table 4.1. A generalized relation

Table 4.1 is a generalized relation with five tuples which implies a rule with five disjuncts. Obviously, further generalization is needed to reduce the number of tuples. In practice, it is often necessary to set up a *threshold*, an upper bound on the number of tuples in the final generalized relation. Suppose our threshold value is set to three in this example. Since only the attribute "Birth_Place" contains four distinct values, generalization should be performed on this attribute by ascending one level in the concept tree resulting the relation shown in Table 4.2.

Major	Birth_Place	GPA
art	Canada	excellent
science	Canada	excellent
science	Foreign	good

Table 4.2. Further generalization of the relation.

Since generalization is controlled by the threshold value, we have

Generalization Strategy 4.4. (Threshold control) *If the number of distinct values in a resulting relation is larger than the specified threshold value, further generalization on this attribute should be performed.*

Reasoning.

If the number of distinct values in a resulting relation is larger than the specified threshold value, it should be generalized. Otherwise the final generalized relation will contain more tuples than the specified threshold value. □

Notice that generalization can be performed on one attribute several times consecutively by ascending several levels up a concept tree without generating intermediate relations. Such localized generalization on one attribute saves processing cost [Win75]. For example, the generalization on "Birth_Place" can be performed twice because the first generalization produces four distinct values, which is still above the threshold value, and it is necessary to perform one additional generalization. Therefore, we obviate the need to generate Table 4.1; only Table 4.2 is generated.

The above processing ensures that the number of distinct values in each attribute of the resulting relation is no larger than the specified threshold value. However, the

total number of tuples in the resulting relation may still be above the threshold value. In this case, further generalization on some attribute should still be performed. The choice of the generalized attribute may depend on the tuple reduction ratio, simplicity of the final learned rules, etc.

Among the three resulting tuples, simplification can be performed by "unioning" the first two tuples if the set representation of an attribute is allowed. Logically, this is equivalent to

$$(x_1 \wedge y \wedge z) \vee (x_2 \wedge y \wedge z) \equiv (x_1 \vee x_2) \wedge y \wedge z. \quad (4.1)$$

Thus we obtain Table 4.3:

Major	Birth_Place	GPA
{ art, science }	Canada	excellent
science	Foreign	good

Table 4.3. Simplification of the generalized relation.

Since art and science cover all of the Major areas, {art, science} can be generalized to ANY and then removed from the representation. Therefore, the final generalized relation is shown in Table 3.3, which is equivalent to rule (*4.1*). That is, *a graduate is either a Canadian with an excellent GPA or a foreign student, majoring in sciences with a good GPA.*

(*4.1*) $\forall$ (x) graduate(x) =>
(Birth_Place(x) $\in$ Canada $\wedge$ GPA(x) $\in$ excellent) $\vee$
(Major(x) $\in$ science $\wedge$ Birth_Place(x) $\in$ Foreign $\wedge$ GPA(x) $\in$ good).

Similarly, if the algorithm is applied to learn rules for "undergraduate students", we will obtain rule (*4.2*), that is, *all undergraduate students are Canadians.*

(4.2) $\forall$ (x) undergraduate(x) => Birth_Place(x) $\in$ Canada.

As shown in rules (4.1) and (4.2), since the learned rules cover all of the data in the learning class, they are the necessary conditions of the learning class. We will discuss this in more detail in Chapter 6.

From this discussion, the learning algorithm, LCHR, can be summarized as follows.

Algorithm 4.1. *LCHR — Learning characteristic rules from relational databases.*

Notation. { } is used to enclose a comment. P is a relation of the database relevant to the learning task, which consists of a set of attributes A_i, $1 \leq i \leq n$, where n is the number of attributes in relation P. N stands for the total number of tuples in the current (working) relation, and d_i is the number of distinct values of attribute A_i of the current relation. T stands for the user-specified threshold value, i.e., the maximum number of disjuncts in the resulting rule.

Input.

(i) a relational database,

(ii) a concept hierarchy table,

(iii) the learning task, and

(iv) the threshold value (T).

Output. A characteristic rule learned from the database.

Method.

Step 1. *Select the task-relevant data, relation P*, using relational operations and

concept hierarchy table when necessary. (The method was discussed in Chapter 3).

Step 2. *Perform attribute-oriented induction,* which is described by the following procedure.

Procedure *Attribute-oriented induction for learning characteristic rules*;

```
{ Generalization is performed as follows on each attribute A_i of P . }
BEGIN
    FOR EACH attribute A_i DO
    BEGIN
        WHILE d_i > T DO
            IF there is no higher level concept in the concept hierarchy table for
                the values of A_i
            THEN remove attribute A_i
            ELSE { There is a higher level concept. }
                Substitute the values by its corresponding minimal generalized
                concept, and
                Eliminate redundant tuples;
    END
    { Now d_i ≤ T }
    WHILE N > T DO
    BEGIN
        Generalize the attributes containing substantially more distinct
            values or those with a better reduction ratio (i.e., reducing to
            a less number of tuples), and
        Eliminate redundant tuples;
            { Now N ≤ T }
    END
END. {Attribute-oriented induction for learning characteristic rules}
```

Step 3. *Simplify the generalized relation.*

If only one attribute of several tuples contains distinct values, the several tuples can be reduced into one by taking the distinct values of that attribute as a set.

Step 4. *Transform the final relation into logic formulas.*

Based on the semantics of relations expressed in logic [GMN84], one tuple is

transformed to a conjunctive normal form, and multiple tuples are transformed to a disjunctive normal form. □

We are now in a position to state the following theorem.

Theorem 4.1. Algorithm LCHR correctly learns characteristic rules from relational databases.

Proof Sketch.

Step 1 collects relevant data in the database for the learning task. Generalization Strategy 4.2 is used in step 2 to ensure that generalization is performed on the least decomposable components. The THEN-part in the first WHILE loop of Step 2 is based on Generalization Strategy 4.1 (removing attributes), and the ELSE-part is based on Generalization Strategy 4.3 (ascending the concept tree). The second WHILE loop in step 2 is based on Generalization Strategy 4.4 (controlled by the threshold value). Each generalization statement in both WHILE loops applies the least-commitment principle based on those strategies. Finally, steps 3 and 4 apply logic transformations based on the correspondence between relational tuples and logic formulas. Thus, the obtained rule should be the desired result which summarizes the characteristics of the class. □

4.2. Learning Classification Rules

Besides the characteristic rules, the classification rules are also very useful in many applications. Since a relational database stores a vast amount of data, it can be viewed as a set of typical samples of the real world. The classification rules derived

from a databases can be used to classify the new data, and predict the properties of the new data according to their class memberships.

We now describe the second database learning algorithm, LCLR, *Learning CLassification Rules from databases* [CCH90]. Similar to the LCHR algorithm, the LCLR algorithms also applies the attribute-oriented induction technique. The difference is that in the extraction of classification rules, the facts which support the target class serve as positive examples, while the facts which support the other classes serve as negative examples. Since the learning task is to discover the concepts that have discriminant properties, the portion of facts in the target class that overlaps with other classes should be detected and removed from the description of classification rules. We analyze such a learning process using another example.

Example 4.1. Learning a classification rule which distinguishes graduate students from undergraduate students in the relation of Table 3.1.

Since all of the classes relevant to the learning task are used in the learning process, it is necessary to extract the data related to those classes. For this learning task, the data in the target class, graduate students, serve as positive examples, and the data in the contrasting class, undergraduate students, serve as negative examples. Clearly, the learning process should be performed on the entire relation in Table 3.1. To facilitate the learning process, the data should be clustered by classes. The attribute "Category" can be removed after the relevant data are selected, which is the same as that in LCHR because this attribute is not related to the learning task afterwards.

Similar to LCHR, this algorithm repeatedly performs generalization by "*ascending the concept tree*" or by "*attribute removal*". After removing two attributes, Name and Category, and generalizing the three remaining attributes, Major, Birth_Place and GPA, the relation depicted in Table 3.1 is generalized to a new relation as illustrated in Table 4.4 (with redundant tuples eliminated). The first five tuples belong to the class graduate-student, and the last six tuples belong to the class undergraduate-student.

Learning Concept	Major	Birth_Place	GPA	Mark
graduate	art	B.C.	excellent	
	science	Ontario	excellent	
	science	B.C.	excellent	*
	science	India	good	
	science	China	good	
undergraduate	science	Alberta	excellent	
	art	Alberta	average	
	science	B.C.	average	
	science	B.C.	excellent	*
	art	B.C.	average	
	art	Ontario	excellent	

Table 4.4. A generalized relation

As shown in Table 4.4, different classes may share tuples. We define *overlapping tuples* as follows.

Definition: A set of *overlapping tuples* is a set of tuples which are shared by different classes.

Obviously, the third tuple of class *graduate-student* and the fourth tuple of class *undergraduate-student* are overlapping tuples, which indicates that a B.C. born student, majoring in science with good GPA, may or may not be a graduate student. Therefore, in order to get an effective classification rule, care must be taken to handle

the overlapping tuples. We have

Generalization Strategy 4.5. (Handling overlapping tuples) *If there are overlapping tuples in both target and contrasting classes, these tuples should be marked and eliminated from the final generalized relation.*

Reasoning.

Since the overlapping tuples represent the same assertions in the target class and the contrasting class, they cannot be used to characterize the distinction of the target class from the contrasting class. By detecting and removing the overlapping tuples, only the assertions which have a discriminating property remain in the classification rule, which guarantees the correctness of the learned rules. Removing a tuple is a specialization process, which is the opposite operation of the generalization rule, *adding options*. Consider multiple-tuples as a disjunction, and each tuple as one of the disjuncts. By removing a disjunct, we eliminate one option and thus specialize the rule. □

After marking the third tuple in the class of graduate-student and the fourth tuple in the class of undergraduate-student, the target class contains four unmarked tuples as shown in Table 4.4, which implies that the resulting rule will contain four disjuncts. Suppose the same threshold value, 3, is specified as before. Then based on the arguments similar to those in LCHR, further generalization is performed on the attribute "Birth_Place", which results in the relation shown in Table 4.5.

Learning Concept	Major	Birth_Place	GPA	Mark
graduate	art	Canada	excellent	*
	science	Canada	excellent	*
	science	Foreign	good	
undergraduate	science	Canada	excellent	*
	arts	Canada	average	
	science	Canada	average	
	art	Canada	excellent	*

Table 4.5. A generalized relation

Notice that the overlapping mark should be inherited in their generalized tuples because the generated concept still overlaps with the concept in other class(es). Moreover, since such generalization may produce new overlapping tuples, overlapping checking should be performed in each ascending of the concept tree. The judgement for further generalization or attribute removal should rely on the unmarked tuples in the target class. The generalization process is repeated until the number of distinct values in each attribute in the unmarked tuples is under the specified threshold value for the target class. Then, if the target class contains more unmarked tuples than the threshold value, further generalization on some selected attribute is still needed, which is similar to LCHR.

After eliminating the marked tuples, only one tuple is left in the target class in the example. Based on the same principles in LCHR, the final generalized relation can be transformed to the corresponding logic formula. The classification rule for "graduates" is rule *(4.3)*: *if a student is from a foreign country, majoring in sciences with a good GPA, he/she is a graduate student.*

(4.3) $\forall$ (x) graduate(x) <=
Major(x) $\in$ science /\ Birth_Place(x) $\in$ Foreign /\ GPA(x) $\in$ good

Similarly, the classification rule for "undergraduates" is rule (*4.4*): *if a student is a Canadian with an average GPA, he/she is an undergraduate student.*

(*4.4*) $\forall$ (x) undergraduate(x) <= Birth_Place(x) $\in$ Canada /\ GPA(x) $\in$ average

In contrast to the LCHR algorithm, this algorithm learns the rule which is the sufficient condition of the learning concept, because the generalized rule excludes the concepts which cover the tuples in other classes. More discussion on this will be presented in Chapter 6.

From the above discussion, LCLR algorithm can be summarized as follows.

Algorithm 4.2. *LCLR — Learning classification rules from relational databases.*

Notation. { } is used to enclose a comment. P is a relation of the database relevant to the learning task, which consists of a set of attributes A_i, $1 \leq i \leq n$, where n is the number of attributes in relation P. N stands for the total number of unmarked tuples in a class, and d_i is the number of distinct values of attribute A_i in unmarked tuples of a class. T stands for the user-specified threshold value, i.e., the maximum number of disjuncts in the resulting rule.

Input.

(i) a relational database,

(ii) a concept hierarchy table,

(iii) the learning task, and

(iv) the threshold value (T).

Output. A classification rule for the target class learned from the database.

Method.

Step 1. *Select the task-relevant data of the target class and the contrasting class to form relation P and cluster the data by classes.*

Step 2. *Perform attribute-oriented induction,* which is described by the following procedure.

Procedure *Attribute-oriented induction for learning classification rules*;

{ Generalization is performed as follows on each attribute A_i of each class. }
BEGIN
    FOR EACH attribute A_i DO
    BEGIN
        Perform intersection of both classes and mark the overlapping tuples;
        WHILE d_i in the target class > T DO
            IF there is no higher level concept in the concept hierarchy table of A_i
            THEN remove attribute A_i;
            ELSE { There is a higher level concept. }
                BEGIN
                    Substitute the values by its corresponding minimal generalized
                        concept (with overlapping marks automatically inherited);
                    Mark the newly generalized tuples which overlap with the tuples
                        in other classes; and
                    Eliminate identical tuples within each class
                END
    END

    { Now, the number of distinct values of each remaining attribute in the
    target class is less than T. }
    WHILE N in the target class > T DO
    BEGIN
        Generalize the attributes containing more distinct values than others
            or those with a better reduction ratio (with overlapping marks
            automatically inherited);
        Mark the newly generalized tuples which overlap with those in the contrasting
            class; and
        Eliminate identical tuples within each class
    END
END. {*Attribute-oriented induction for learning classification rules*}

Step 3. *Remove overlapping tuples and simplify the generalized relation.*

In this step, the marked tuples are first eliminated; then the simplification process is the same as that of LCHR.

Step 4. *Transform the final relation into logic formulas.*

This step is similar to Step 4 in LCHR except that the resulting formula is a sufficient condition of the learning concept.

Theorem 4.2. Algorithm LCLR correctly learns classification rules from relational databases.

Proof Sketch.

Step 1 collects the relevant data in the database for the learning task. Step 2 generalizes the concept in each attribute either by "*ascending the concept tree*" (Generalization Strategy 4.3) or by "*attribute removal*" (Generalization Strategy 4.1), which simulates the generalization process of *learning from examples*. Moreover, the specified threshold value ensures that the process of ascending of the tree terminates when it reaches the threshold-controlled number of disjunctions (Generalization Strategy 4.4), and "*removing overlapping tuples*" guarantees the resulting properties are not shared by the contrasting class (Generalization Strategy 4.5). Step 3 and Step 4 perform simplification and transformation based on logic transformation rules. Thus, the obtained rule should be the desired result which characterizes the discriminating property of the class. □

CHAPTER 5

VARIATIONS OF THE LEARNING ALGORITHMS

In Chapter 4 we presented a general outline for learning characteristic rules and classification rules from relational databases and the LCHR and the LCLR learning algorithms. We now discuss some variations of these algorithms which can cope with different learning situations.

5.1. Adjusting Thresholds for Different Learning Results

Although there are some learning algorithms which can learn disjunctive rules [MiC80, MMH86], many learning algorithms learn only conjunctive rules [HaM77, HaM78, MCM86, Mit77, Mit82, Ver75]. By permitting the threshold value (the maximum number of disjunctions in the resulting formula) to be a small integer, the LCHR algorithm and the LCLR algorithm can learn both conjunctive and disjunctive rules. Such flexibility facilitates learning in many applications.

Algorithms LCHR and LCLR require the specification of threshold values by users. There could be other variations, such as predefining the threshold value by a database administrator or an expert. A threshold value, T, is usually small. There is a tradeoff between small versus moderately large threshold values. A moderately large threshold value may lead to relatively complex rule, containing many disjuncts and some half-generalized results. A small threshold value leads to a small final relation, that is, a simple rule with few disjuncts. However, it may result in an over-generalized rule, and some valuable information may be lost, as the following

example illustrates.

Example 5.1. Different threshold values result in different learning results.

Suppose we have a generalized relation depicted in Table 5.1.

Major	Birth_Place	GPA
art	Canada	average
science	Canada	excellent
science	Foreign	good

Table 5.1. A generalized relation.

If the threshold value is set to 3, this generalized relation is the learning result. However, if 2 is the threshold value, the attribute "GPA" should be further generalized, which will give the learning result shown in Table 5.2.

Major	Birth_Place	GPA
{art, science}	Canada	ANY
science	Foreign	ANY

Table 5.2. The learning result with the threshold value 2.

The appropriate threshold value varies in different learning situations [Fis88]. It is not possible to define a uniform threshold value that is suitable for any learning task. A better way to determine a threshold value is to adjust the threshold values within a reasonable range in several tests and then examine the learning results and select the best one by consultation with domain experts and users.

The final relation resulting from LCHR algorithm and LCLR algorithm may consist of a small set of disjuncts in which each disjunct is a set of conjuncts. The advantage of our approach is that we learn some disjunctive rules and still keep the number of disjuncts small. Note that some other algorithms such as [MiC80] can

learn such rules as well. If disjunctive forms were allowed at the early generalization stage, many concepts may never be generalized because they can be represented by disjunctive forms. Our approach restricts disjunct construction to a later stage. A relaxation to this method is to allow some disjuncts in intermediate relations, which makes the generalization process more conservative, and thus avoids some possible over-generalization.

5.2. Dealing with Different Kinds of Concept Hierarchies

In our examples, all of the concept hierarchies are represented as the balanced concept trees and all of the primitive concepts reside at the same level of a concept tree. Hence generalization can be performed synchronously on each attribute to generalize the attribute values at the same lower level to the ones at the same higher level. However, we may encounter other kinds of concept hierarchies or we may encounter the case where the primitive concepts do not reside at the same level of a concept tree.

5.2.1. Generalization of the Concepts at Different Levels of a Hierarchy

The concept hierarchies may be organized as unbalanced concept trees. For example, the left branch of a tree may have fewer levels to the leaves than the right branch. In these cases, synchronous tree ascension may reach the same level at different stages, which may result in an incorrect generalization at that level. A similar problem may occur when the primitive concepts reside at the different levels of a concept tree. These problems can be solved by checking whether one generalized concept may cover other concepts of the same attribute. If one generalized concept

covers a concept several levels down the concept tree, the covered concept is then substituted for by the generalized concept, that is, ascending the tree several levels at once. In doing so, concepts at different levels can be handled correctly and efficiently.

Example 5.2. Handling an unbalanced concept tree.

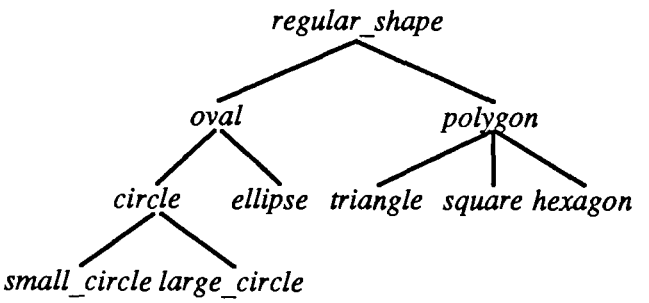


Figure 5.1. An unbalanced concept tree.

Figure 5.1 shows an unbalanced concept tree. Based on the discussion above, as long as the attribute value "ellipse" has been generalized to "oval", those attribute values, "small_circle", "large_circle" and "circle", can be substituted by "oval" at once.

This idea can be used for incremental learning as well. Relational databases are characterized by frequent updating. As new data become available, it will be more efficient to amend and reinforce what was learned from previous data than to restart the learning process from scratch [KuS88, MMH86]. Our algorithms can be easily extended to perform incremental learning. When new data are presented to a database, an efficient approach to characterization and classification of data is to first

generalize the concepts of the new data up to the level of the rules which have been learned, then the LCHR and LCLR algorithms can be used to merge the generalized concepts derived from the old data and the new data, which is illustrated in the following example.

Example 5.3. Incremental learning when new data are inserted into the database.

Suppose Table 5.3 is the characteristic rule for graduate students derived from the original data in the database.

Major	Birth_Place	GPA
science	Canada	excellent
science	Foreign	good

Table 5.3. The generalized relation induced from the original database.

Suppose the new data of Table 5.4 are inserted into the database.

Name	Category	Major	Birth_Place	GPA
Bent	M.S.	music	Ottawa	3.2
Gale	M.S.	history	Burnaby	3.3
Lewis	Ph.D.	math	Vancouver	3.8
Tomps	M.S.	biology	Toronto	3.7

Table 5.4. The new inserted data.

Instead of performing learning on the updated database from scratch, incremental learning can be performed by first generalizing the new data to the level of the rule presented in Table 5.3, that derives Table 5.5.

Major	Birth_Place	GPA
art	Canada	good
science	Canada	excellent

Table 5.5. The generalized relation for new data.

Then merging the newly derived generalized relation (Table 5.5) and the old one (Table 5.3). Suppose the threshold value is 3, then the merged generalized relation table is Table 5.6 which is the new characteristic rule for graduate students.

Major	Birth_Place	GPA
science	Canada	excellent
science	Foreign	good
art	Canada	good

Table 5.6. The new characteristic rule for graduate students.

Such incremental learning significantly saves computational cost, especially when the size of the database is large.

5.2.2. Generalization of Concepts in the Hierarchies with Lattices

In all of our previous examples, the concept hierarchies are trees, that is, every node has only one parent node at most. For any concept, therefore, there is only one direction to perform the generalization. In some cases, however, the concept hierarchy may have lattice(s). Figure 5.2 illustrates this case.

Example 5.4. Handling a concept hierarchy with lattices.

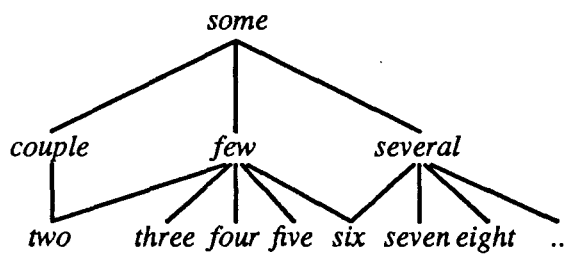


Figure 5.2. A concept hierarchy with lattices.

Clearly, the concept "two" can be generalized either to *couple* or *few*. Both generalized concepts should be considered. Our method is to put all possible generalized concepts into intermediate generalized relations when a lattice is encountered, and then perform further generalization on all those tuples. In this example, after the tuple containing attribute value "two" is generalized, two new tuples, containing attribute values "couple" and "few", respectively, should be generated. For the concept "six", the same technique should be applied. As a consequence, the size of the generalized relation table may increase at some stage of the generalization process because of the effect of a lattice. However, since the generalization is controlled by the specified threshold value, the generalized relation will eventually shrink in further generalizations.

5.3. Nonuniqueness of Learning Results

When a relation is generalized close to the final stage, interesting rules can often be discovered by generalization in several possible directions. For example, when the number of distinct values in each attribute has been reduced to below the threshold value, there could be several choices to select the attribute on which further generalization should be performed. Taking different attributes to perform further generalization may result in different learning results. Example 5.5 shows the possibility of nonuniqueness of the learning results.

Example 5.5. Nonuniqueness of rules learned from the same data.

Suppose Table 5.7 is an intermediate generalized relation for learning a characteristic rule of students.

Major	Birth_Place	GPA
art	B.C.	good
science	B.C.	good
science	B.C.	excellent
art	Ontario	good
science	Ontario	good

Table 5.7. An intermediate generalized relation.

Since Table 5.7 contains five tuples, which is beyond the specified threshold value of 3, further generalization is needed. However, there are 2 choices on the attributes to perform further generalization because the attribute "Major" and the attribute "Birth_Place" have the same tuple reduction ratio. By generalization on the attribute "Major", Table 5.8 can be derived.

Major	Birth_Place	GPA
Any	B.C.	{good, excellent}
Any	Ontario	good

Table 5.8. A possible learning result.

This table corresponds to the rule, *all of the students from the province of B.C. have good or excellent GPAs, and all of the students from the province of Ontario have good GPAs.*

On the other hand, if the further generalization is performed on the attribute "Birth_Place", Table 5.9 should be the result.

Major	Birth_Place	GPA
art	Canada	good
science	Canada	{good, excellent}

Table 5.9. Another possible learning result

This rule represents a different concept from the previous one, that is, *all students majoring in art have good GPAs, and all students majoring in science have good or excellent GPAs.*

In such a situation, it is often desirable to perform generalizations in several directions to obtain several final generalized relations, which corresponds to the fact that different people learn differently from the same examples. The final generalized relations should be examined by users or experts to filter out some trivial generalizations and preserve interesting results.

5.4. Incorporating Quantitative Information

Both characteristic rules and classification rules learned by LCHR algorithm and LCLR algorithm represent qualitative knowledge rules which do not provide and utilize any quantitative information. These algorithms can be easily extended to discover additional quantitative information which can be used to provide quantitative evaluation, as well as handle noisy data and exceptional cases.

The technique we adopt is to add a special attibute, *Votes*, to each generalized relation. This attribute registers the number of tuples in the original relation which are generalized to one tuple in the current generalized relation. For example, a generalized tuple associated with the *Votes* value 30 indicates that this tuple is derived from 30 tuples in the original relation. Based on the values in the attribute *Votes*, we can calculate two weights, *t-weight* and *d-weight*, where *t* stands for *typical*, and *d* stands for *discriminant.*

Definition: Let q_a be a generalized concept. The t-weight for q_a is the percentage of

tuples covered by q_a in a class.

Formally, the t-weight for concept q_a can be defined as follows.

$$t\text{-}weight = \frac{Votes(q_a)}{\sum_{i=1}^{N} Votes(q_i)} \tag{5.1}$$

where N is the number of tuples in the final generalized relation, and q_a is in $\{q_1 .. q_N\}$.

The range of values for t-weight is [0, 1]. This weight may be interpreted as a measure of the typicality or the representativeness of a generalized tuple as the characteristic rule of the data.

Definition: Let q_a be a generalized concept, and C_j be a class. The d-weight for q_a in C_j is the ratio of the number of tuples in C_j covered by q_a to the total number of tuples in all of the classes covered by q_a.

The following formula is used to calculate the d-weight of the concept q_a in class C_j:

$$d\text{-}weight = \frac{Votes(q_a \text{ in } C_j)}{\sum_{i=1}^{K} Votes(q_a \text{ in } C_i)} \tag{5.2}$$

where K stands for the total number of the classes, and C_j is in $\{C_1 .. C_K\}$.

The range of values for d-weight is [0, 1]. A high d-weight indicates that the concept is mainly derived from the current class, and a low d-weight implies that the concept is shared by other class(es).

5.4.1. Association of Quantitative Information in the Induction Process

By calculating the *t-weight* when learning characteristic rules, and calculating *t-weight* and *d-weight* when learning classification rules, the learning process can be augmented with quantitative evaluation. We present some examples to illustrate these ideas.

Example 5.6. Calculating the t-weight for a characteristic rule.

Suppose the following final generalized relational table is derived as the characteristic rule of students.

Major	Birth_Place	GPA	Votes
art	Canada	good	30
science	Canada	excellent	25
science	Canada	average	35

Table 5.10. An example of a characteristic rule.

We can discover additional quantitative information by calculating the t-weight. The t-weight for the first tuple in Table 5.10 can be calculated as follows.

$$30 / (30 + 25 + 35) = 33\%$$

Similarly, the t-weight for second and third tuples are 28% and 39%, respectively. Then we conclude that all of the students are Canadian students, among which 33% of the students major in art with good GPAs, 28% of the students major in science with excellent GPAs, and 39% of the students major in science with average GPAs.

Example 5.7. Calculating the t-weight and the d-weight for a classification rule.

Suppose we obtain the following final generalized relation for the classification rule of graduate students.

Learning Concept	Major	Birth_Place	GPA	Mark	Votes
graduate	science	Canada	excellent	*	65
	art	Canada	good	*	15
	science	Foreign	good	*	45
undergraduate	science	Canada	excellent	*	65
	art	Canada	good	*	60
	science	Foreign	good	*	10

Table 5.11. An example of a classification rule.

Since all of the generalized concepts in these two classes are overlapping tuples, no classification rule can be learned if the original algorithm is applied. However, we still can discover valuable information if we calculate the t-weight and the d-weight.

Similar to the calculation in the last example, by applying the formula (5.1), we obtain the information that 52% graduate students are from Canada, majoring in science with excellent GPAs, 12% graduate students are Canadian and majors in art with good GPAs, and 36% graduate students come from foreign countries and major in science with good GPAs.

Based on formula (5.2), the d-weight for the first tuple in the target class *graduate* is calculated as follows.

$$65 / (65 + 65) = 50\%$$

We can conclude that if a student is from Canada and majors in science with an excellent GPA, he/she is a graduate student with a frequency of 50%. Similarly, when a student has the conditions in the second tuple or the third tuple, the frequency that the student is a graduate student are 20% or 82%, respectively. Note that if a generalized tuple is not shared with other classes, its d-weight will be 100%.

In the real world, the instances of different classes will often be overlapped. By calculating the d-weight, important information implied in this kind of data can be discovered. The learned rules provide quantitative criteria to determine the class membership of new data.

5.4.2. Handling Noisy Data and Exceptional Cases

Many learning systems have been developed under the assumption that there exists no noise in the training examples. As a consequence of this assumption, careful data gathering is required to ensure that the training examples are always paired with their correct classification and that their descriptions are perfectly reliable in the sense that they are error-free [MaK87, WoC88]. In real applications, however, a learning system has to operate in an enviroment where there are different sources of uncertainty:

(1) incorrect, inconsistent and inaccurate values may be present;

(2) unusual, less representative values may be included;

(3) misclassified values may exist.

Many researchers consider a small number of unusual cases as noisy data, and many techniques have been developed to cope with noisy data [MaK87, Qui86, WoC88]. By incorporating quantitative information and using statistical techniques, our algorithms can be extended to handle noisy data and exceptional cases presented in the relational database.

Since the *Votes* of a generalized tuple indicates the number of tuples that it is generalized from, and t-weights are derived from *Votes*, t-weights carry the statistical

information of a database. A high t-weight implies that the concept is induced from a majority of data, and a very low t-weight implies that the concept is derived from some rare, exceptional cases. Therefore, a high t-weighted tuple should be preserved in the generalized relation while a very low t-weighted tuple should be removed from the generalized relation if the goal is to learn the characteristics of a majority number of tuples. By doing so, the final generalized rule will characterize the majority of the facts in the database. Example 5.8 illustrates this idea.

Example 5.8. Pruning noisy data and exceptional cases from the generalized relation in Table 5.12.

Major	Birth_Place	GPA	Votes
art	Canada	average	1
science	Canada	excellent	64
science	Foreign	good	35

Table 5.12. A generalized relation with an exceptional case.

Obviously, the tuples in this generalized relation carry different t-weights, which indicates that some concepts come from the majority of data, but some do not. In general, we may specify a *pruning threshold* in the learning process. If the pruning threshold value is set to 5%, the first tuple should be dropped since its t-weight is 1%. Then we conclude that 99% of students are science students with GPA ranging from good to excellent.

As discussed in the presentation of the LCLR algorithm, there could be some overlapping tuples discovered in learning classification rules. Some of these overlapping tuples may come from an incorrect classification, and some may belong to some

exceptional case(s). By incorporating some statistical techniques, these kinds of noisy data and exceptional cases can be detected. We adopt a method that measures the correlation coefficient between a class and a generalized tuple [ImC83, KKM88, Pia89].

We define *association ratio*, ϕ, as follows.

Definition: Let C be a class, and Q be a generalized tuple. The *association ratio*, ϕ, is the correlation coefficient between C and Q calculated by the following formula.

$$\phi = \frac{|C\&Q| - |C||Q|/N}{[|C||Q|(1-|C|/N)(1-|Q|/N)]^{1/2}} \tag{5.3}$$

where N is the total number of tuples in the original task-relevant relation, $|C|$ is the total number of votes in the class C, $|Q|$ is the number of votes of a generalized tuple Q in all classes, and $|C\&Q|$ is the number of votes of Q in class C.

Measuring the association ratio is a standard statistical problem for 2 * 2 contingency tables. This formula is derived from the formula (5.4) for calculating the *sample correlation coefficient.*

$$r = \frac{\sum_{i=1}^{n} X_i Y_i - (\sum_{i=1}^{n} X_i)(\sum_{i=1}^{n} Y_i)/n}{[[\sum_{i=1}^{n} X_i^2 - (\sum_{i=1}^{n} X_i)^2/n][\sum_{i=1}^{n} Y_i^2 - (\sum_{i=1}^{n} Y_i)^2/n]]^{1/2}} \tag{5.4}$$

which is approximately normally distributed [ImC83, KKM88, Pia89].

The possible values of association ratio ϕ range from -1 to 1, which can be interpreted as the significance of association between C and Q. The more positive ϕ is, the stronger is the association. This means that when ϕ is close to 1, high $|C|$ will

be likely associated with high $|Q|$, and low $|C|$ will be likely associated with low $|Q|$. The more negative ϕ is, the more negative is the association; that is, high $|C|$ may associate with low $|Q|$ when ϕ is close to -1, and vice versa. In general, a very low ϕ indicates that the generalized tuple Q is misplaced in the class C, or belongs to the exceptional cases in the class C.

A threshold value, *significance threshold*, can be specified by users or experts. If the association ratio ϕ for C and Q is below the significance threshold, the generalized tuple Q should be removed from the class C. We use Example 5.9 to show this method.

Example 5.9. Measurement of the association ratio between a class and a generalized tuple in Table 5.13.

Learning Concept	Major	Birth_Place	GPA	Mark	Votes
graduate	science	Canada	excellent	*	40
	art	Canada	excellent		12
	science	Foreign	good	*	48
undergraduate	science	Canada	excellent	*	130
	art	Canada	good		68
	science	Foreign	good	*	2

Table 5.13. A generalized relation for a classification rule.

By calculating the association ratio ϕ for the third tuple in Table 5.13, we can determine whether this tuple is significantly correlated with the class *graduate*. Clearly, we have

$$|C\&Q| = 48,$$
$$|C| = 40 + 12 + 48 = 100,$$
$$|Q| = 48 + 2 = 50,$$
$$N = 40 + 12 + 48 + 130 + 68 + 2 = 300,$$

$$\phi = \frac{48 - 100 * 50 / 300}{[100 * 50 * (1 - 100 / 300)(1 - 50 / 300)]^{1/2}} = 0.59$$

Suppose the significance threshold value is set to 0.5. Then by this calculation, we can conclude that the concept represented by the third tuple is significantly correlated with the class *graduate*.

The sixth tuple in the Table 5.13 represents the case of another extreme.

$|C\&Q| = 2,$
$|C| = 130 + 68 + 2 = 200,$
$|Q| = 48 + 2 = 50,$
$N = 40 + 12 + 48 + 130 + 68 + 2 = 300,$

$$\phi = \frac{2 - 200 * 50 / 300}{[200 * 50 * (1 - 200 / 300)(1 - 50 / 300)]^{1/2}} = - 0.59$$

Since the ϕ value is smaller than the significance threshold value, we can conjecture that this generalized tuple is possibly derived from some exceptional cases or from the data which are misclassified. After this tuple is excluded, the third tuple in the class *graduate* becomes the unmarked tuple which should be included in the learned rule.

We have studied several variations of the database learning algorithms in this chapter. With these variations, our learning method can be extended to learning in different situations. Further discussions on the two learning algorithms will be presented in the next chapter.

CHAPTER 6

DISCUSSION

We study the relationship between LCHR algorithm and LCLR algorithm, compare our algorithms with other *learning from examples* algorithms, and discuss the automatic discovery of conceptual hierarchies.

6.1. Necessary Condition versus Sufficient Condition

We have developed two interesting algorithms, LCHR and LCLR, for learning from databases. Both algorithms are attribute-oriented data-driven algorithms which begin with a large number of data and perform generalization, attribute by attribute, and step by step, without referring to a fixed model.

The first algorithm, LCHR, takes task relevant tuples as *positive* examples and adopts the least commitment principle (commitment to minimally generalized concepts), ascending the concept tree only when necessary. Since the generalized rule covers all of the positive examples in the database, it forms the necessary condition of the learning concept. That is, the rule is in the form of

learning_class(x) => condition(x)

where condition(x) is the disjunctive or conjunctive formula containing x. The condition must hold for all the examples of the database in the learning class. However, since data in other classes are not taken as negative examples in the learning process, there could be data in other classes which also meet the specified condition. There-

fore, the learned rule is not a sufficient condition but a necessary condition of the learning class.

The second algorithm, LCLR, treats the tuples of the learning class as *positive* examples and tuples of the contrasting class(es) as *negative* examples. Nevertheless, it adopts the least commitment principle by ascending the concept tree only when necessary. Figure 6.1 illustrates this idea schematically. The dots represent the tuples in one class, and the stars the tuples in another class. Since the generalized rule excludes the generalized concepts which cover the tuples in the contrasting classes, the rule distinguishes the target class from the contrasting classes. However, the generalization may not cover *all* of the positive examples of the target class in the database. Therefore, the learned rule is a sufficient condition of the learning concept but may not be the necessary condition of the learning concept. The rule should be in the form of

learning_class(x) <= condition(x).

That is, if it meets the specified condition, it must be in the target class.

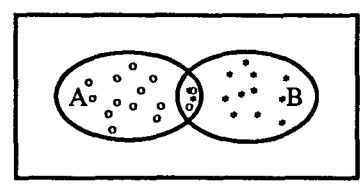


Figure 6.1. Problem space in learning classification rules

As a special case in which there are no overlapping data discovered (marked) in the learning process by the LCLR algorithm, the learned rule represents both

necessary and sufficient conditions of the target class because it covers all of the examples in the target class but no examples in the contrasting classes. The rule is of the form

learning_class(x) <=> condition(x).

6.2. A Comparison with Other Learning Algorithms

Our approach has many distinct features when compared with other learning algorithms.

6.2.1. The Positiveness of the Learning Examples

Many learning from examples algorithms perform learning from both positive examples and negative examples [DiM83, Mic83]. The *candidate elimination* algorithm, the *AQ 11* algorithm and the *AQ 15* algorithm are typical learning systems which follow this paradigm [DiM81, MMH86, Mit77]. In the learning process, both types of examples are necessary and play different roles. The positive examples are used for generalization, and the negative examples are used for specialization. However, since negative examples are not stored in relational databases explicitly, our approach mainly relies on the generalization process. In order to avoid over-generalization, our approach adopts the least commitment principle and threshold control. Negative data are never used in the LCHR algorithm because learning characteristic rules does not have to incorporate such data. In the LCLR algorithm, the data in the contrasting classes are used to exclude the features of the target class which are shared by the contrasting classes. Since the data in the target class and the

contrasting classes are examined simultaneously by an attribute-oriented approach, this specialization process is different from that of *learning from examples*.

6.2.2. Search Space

A concept tree ascending technique is the major generalization technique used in both attribute-oriented generalization and tuple-oriented generalization. However, the tuple-oriented approach performs generalization tuple by tuple, but the attribute-oriented approach performs generalization attribute by attribute. We compare the search spaces of our algorithms with that of a typical method of *learning from examples*, the *candidate elimination* algorithm [CoF83, GeN87, Mit77, Mit82].

In the candidate elimination algorithm, the set of all concepts which are consistent with the training examples is called the *version space* of the training examples. The learning process is the search in this version space to induce a generalized concept which is satisfied by all of the positive examples and none of the negative examples.

Since generalization in an attribute-oriented approach is performed on individual attributes, a concept hierarchy of each attribute can be treated as a factored version space. Factoring the version space may significantly improve the computational efficiency. Suppose there are p nodes in each concept tree and there are k concept trees (attributes) in the relation, the total size of k factorized version spaces is pk. However, the size of the unfactorized version space for the same concept tree should be p^k [SuF86]. This can be verified by Example 6.1.

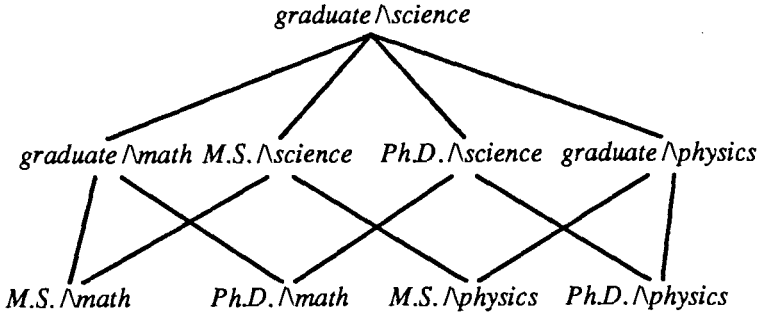
Example 6.1. The entire version space and the factored version space for the concept

hierachy specified in Figure 6.2.

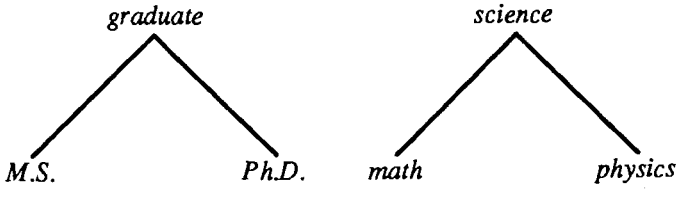
{ math, physics } ⊂ science
{ M.S., Ph.D. } ⊂ graduate

Figure 6.2. A concept hierarchy table.

The corresponding entire version space and factored version space are Figure 6.3a and Figure 6.3b, respectively.



a) The entire version space.



b) The factored version spaces.

Figure 6.3. The entire and factored version spaces for the same concepts.

Obviously, the entire version space contains $3^2 = 9$ nodes, but the factored version spaces contain a total of $3 * 2 = 6$ nodes. The search space for our algorithms is much smaller than the one for the candidate elimination algorithm.

6.2.3. Conjunctive Rules and Disjunctive Rules

Many machine learning algorithms, such as Winston's algorithm for learning concepts about the blocks world [Win75], the *candidate elimination* algorithm [Mit77, Mit82] which is concerned with the discovery of a conjunctive rule when both positive examples and negative examples are presented, program *Thoth* [Ver75] and program *SPROUTER* [HaM77, HaM78] both of which are designed for finding the maximally-specific conjunctive generalizations of a set of input positive examples, can only learn conjunctive rules [CoF83]. The goal of the learning process performed by such algorithms is to induce a conjunctive rule which can be satisfied by all training examples. Since disjunctions allow a partially ordered rule space to become infinitely "branchy", many algorithms do not permit disjunctions in the representation language. However, in real world applications, there are many knowledge rules which should be expressed in a disjunctive normal form. Our algorithms can learn both conjunctive and disjunctive rules under the control of a specified threshold value. If the threshold value is set to 1, the learning result will be a conjunctive rule. Otherwise, if the threshold value is a small integer greater than 1, the learning result will be a disjunctive rule consisting of a small number of conjuncts. Our approach provides additional flexibility over many machine learning algorithms. Moreover, by adjusting thresholds in the learning process, our approach can learn knowledge rules in different conjunctive and disjunctive forms. This learning process provides more choices for experts and users to select the more desirable ones.

6.2.4. Handling Overlapping Instances

We compare our method to another interesting learning program AQ11, which is designed for learning a set of classification rules [MiC80]. AQ11 converts the problem of learning classification rules into a series of single-concept learning problems. To find a rule for class A, it considers all of the known instances in class A as positive instances and all the training instances in the remaining classes as negative instances. The algorithm of AQ11 is then applied to find a concept that covers all of the positive instances without covering any of the negative instances. This program works only when there exist no overlapping instances among different classes. Our algorithm can learn classification rules regardless of the existence of overlapping instances. The overlapping instances can be detected during the learning process and removed in the final learning step, which guarantees the discriminant property of the learned rules. Thus our methods apply to more real world problems.

6.2.5. Utilizing Database Facilities

Relational database systems provide many attractive features for machine learning, such as the capacity to store a large amount of information in a structured and organized manner and the availability of well developed implemetation techniques. However, most existing algorithms do not take advantage of these database facilities [DiM83, KMK89, Mic83, Pia89]. An obvious advantage of our approach over many other learning algorithms is the integration of the learning process with database operations. Most of the operations used in our approach involve traditional relational database operations, such as selection, join, projection (extracting relevant data and

removing attributes), tuple substitution (ascending concept trees), and intersection (discovering common tuples among classes). These operations are set-oriented and have been efficiently implemented in many relational systems. While most learning algorithms suffer from inefficiency problems in a large database environment [DiM83, Mic83], our approach can use database facilities to improve the performance.

Moreover, in contrast to many machine learning algorithms which can learn only qualitative rules, our approach can learn qualitative rules with quantitative information. Some learning systems can only work in a "noise free" environment [MaK87, WoC88], but our approach can handle noisy data and exceptional cases elegantly by the incorporating statistical techniques in the learning process. With these features, our approach provides a simple and efficient way to learn knowledge rules from large databases.

6.2.6. Limitations of the LCHR and LCLR Algorithms

The LCHR and LCLR algorithms are designed for learning characteristic rules and classification rules from relational databases. Therefore, both algorithms can only handle the well-formatted data stored in relational databases. The databases of many applications may contain complex data objects which may not be in the first normal form, that is, an attribute of a tuple could contain structures or set values or be represented in disjunctive forms [KoS86]. The handling of such nonrelational data is beyond the capability of our algorithms. This is one of the major limitations of our method.

Many other learning algorithms learn complex formatted data which may not be in the first normal form of the relational databases [HaM77, Mic83, MMH86, Ver75, Win75]. It is an interesting research issue to extend our technique to complex data objects.

6.3. Discovery of Concept Hierarchies

One of the key input components to the LCHR and LCLR algorithms is the concept hierarchy table (Figure 3.1), which organizes different levels of abstractions relevant to each attribute of a relation. This table is assumed to be given and to be in the form of concept trees in our algorithm. Now we examine how to discover concept hierarchies.

Method 1: Concept hierarchies are provided by domain experts.

It is realistic to expect that the information about some concept hierarchy is provided by domain experts. Although a database may be large, a concept tree is generally simple and small, and can be input by domain experts in the form of an IS-A hierarchy. A friendly user-interface can be built to facilitate users or domain experts to input the concept hierarchies.

Method 2: Hierarchical information is stored in other portion of databases.

Actually, information about some generalization hierarchies is often be stored in the database. Some data may imply concept hierarchies of other data in the database. For example, "*Vancouver is a city of British Columbia, which is in turn a province of Canada*", may not be explicitly stored in the attribute "Birth_Place". However, it is often stored in a relation about the "Province" and "Country" of each city. If it is

stored in the same relation, the city portion of the "Birth_Place" attribute can be simply eliminated in the generalization process. Otherwise, the relation which stores such information should be retrieved, and the lower level concepts should be substituted by their higher level correspondents. Such a generalization relationship is often specified at the schema level instead of at the tuple level. For example, by indicating the relationship of the attributes in the schema, "city $\subset$ province $\subset$ country", the taxonomy of all the cities stored in the relation are implicitly specified and can be used in the learning process.

Method 3: Concept hierarchies are generated by statistics.

Numerical attributes can be organized in the form of discrete hierarchies to facilitate ascension of concept trees and the representation of learning results [MiS83, Ste87]. Besides explicit specifications of concept hierarchies for numerical values, concept trees can often be built automatically, based on database statistics. Such automatic construction of concept trees can be performed by first obtaining the distribution of attribute values in the database, then setting the range of the values and considering more detailed classifications in more clustered subranges. For example, for an attribute "GPA", suppose that an examination of the values in the database discloses that GPA is between 0 to 4 and most GPAs for graduates are clustered between 3 and 4. One may classify 0 to 1.99 as one class, and 2 to 2.99 as another but give more detailed classification for those between 3 and 4. Since the information is extracted from database statistics, such a concept hierarchy can be constructed automatically.

Even for attributes with discrete values, statistical techniques can be performed under certain circumstances. For example, if the birth-places of most employees are clustered in Canada and scattered in many different countries, the highest-level concepts of the attribute can be categorized as "Canada" and "Foreign".

Method 4: Concept hierarchies are formed by conceptual clustering.

Some learning systems have been developed to organize the observed concepts into a hierarchy of classes. The *HUATUO* system [ChF85] and the *COBWEB* system [Fis87] are typical models in this category. The idea of the technique they adopt is to measure the similarity of the concepts and form the hierarchy of classes based on this measurement. The generated hierachy is a collection of concepts whose intra-class similarity is high and inter-class similarity is low. A similar idea can be applied to generate the concept hierarchies for our learning system.

For numeric attributes, the values which are close to each other can be grouped together. The values can be first partitioned into several classes based on the similarity measure, then following the same principle, each class can be further partioned into sub-classes and so on. For example, the following set of values

{1, 2, 3, 4, 5, 11, 12, 13, 14, 15, 101, 102, 103, 104, 105, 111, 112, 113, 114, 115}

can be organized as the concept hierarchy in Figure 6.4.

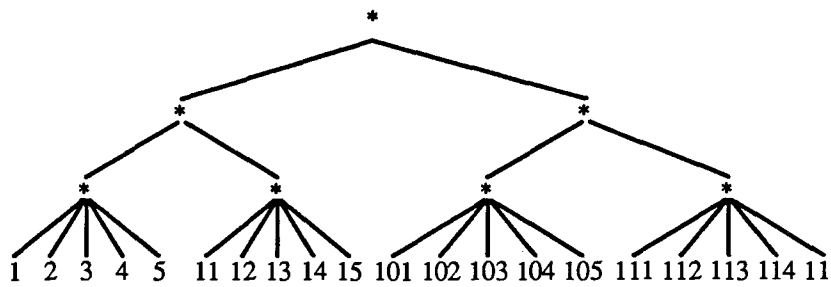


Figure 6.4. A concept hierarchy for a set of numeric value.

The symbols on the root and the intermediate nodes of the hierarchy can be either the symbols automatically generated by the system or the concepts named by domain experts.

For discrete attributes, a similar technique can be applied if the discrete values can be first mapped to the numeric values using a certain mapping technique and then conceptual clustering can be formed based on such mappings [ChF85].

This thesis is focused on the development of efficient database learning algorithms. Further study on the automatic discovery of concept hierarchies is left for future research.

CHAPTER 7

IMPLEMENTATION AND EXPERIMENTS

To test and experiment on the database learning algorithms developed in the previous chapters, an experimental database learning system, **DBLEARN**, has been constructed and some interesting experiments have been conducted in the learning system.

7.1. Implementation of the Database Learning Algorithms

DBLEARN is implemented in C and runs under Unix on a Sun workstation. It implements both the LCHR and LCLR algorithms. The architecture of DBLEARN is presented in Figure 7.1.

In the learning process, DBLEARN first accepts the user's request through the user-interface. Based on the specified learning task, DBLEARN obtains the relevant data from a database and relevant conceptual bias from the file which stores conceptual bias information. One module of the learning program, either LCHR or LCLR, is then invoked based on the user's learning request. After learning is performed, the learning result is reported to the user through the user-interface. Some experiments have been conducted in DBLEARN, which shows great promise of the learning system.

A user-friendly interface is built in the DBLEARN system, by which users can specify the learning task (either the characteristic rule, or the classification rule), the

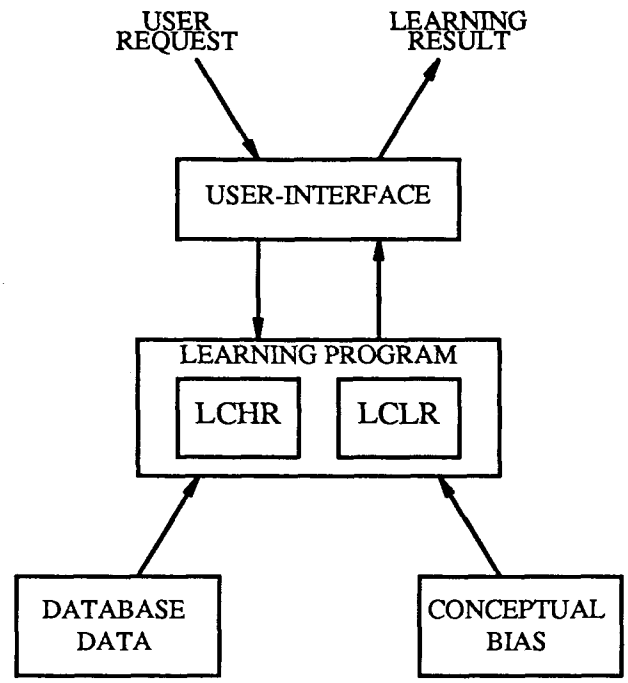


Figure 7.1. The architecture of DBLEARN.

threshold value, the relation and the attributes relevant to the learning task, and the concept to be learned (target class) and the concept to be compared (contrasting class). This interface is implemented using the facilities LEX and YACC on the UNIX system. The syntax of the language is specified in Figure 7.2 using the extended BNF, where { ... } denotes one or more occurrences, *Class_Name, Target_Class_Name, Contrasting_Class_Name, Rel_Name, Attr_Name* are the corresponding names specified by users, and *Int_Val* is a constant greater than 0.

<DBLEARN> : *learn* <rule_type>
<rule_type> : <charact_rule> | <class_rule>
<charact_rule> : *characteristic_rule for* Class_Name <DB_name>
<attr_list> <threshold>
<class_rule> : *classification_rule for* Target_Class_Name *vs*
{ Contrasting_Class_Name } <DB_name> <attr_list> <threshold>
<DB_name> : *from relation* { Rel_Name }
<attr_list> : *relevant to attributes* <attrs>
<attrs> : <attrs> , <attr>
<attr> : Attr_Name
<threshold> : *with threshold* = Int_Val

Figure 7.2. Syntactic specification of DBLEARN.

The following is a sample learning-request which specifies that the task is to learn a characteristic rule for undergraduate students, referring to the attributes Category, Major, Birth_Place and GPA, and the threshold is set to 3.

learn characteristic rule for undergraduate
from relation student
relevant to attributes Category, Major, Birth_Place, GPA
with threshold = 3

If the task is to learn a classification rule for undergraduate students versus graduate students with the threshold value 3 and relevant to the attributes Category, Major, Birth_Place and GPA, the query should be written as follows.

learn classification rule for undergraduate *vs* graduate
from relation student
relevant to attributes Category, Major, Birth_Place, GPA
with threshold = 3

Based on the learning task, the relevant data can be retrieved from a relational database. Usually, the names of the target class and the contrasting class(es) are not the primitive concepts stored in the relation table. For instance, in Example 3.1 and

Example 4.1, the concepts "graduate student" and "undergraduate student" cannot be found in the data. Therefore, it is often necessary to consult the conceptual bias to obtain a set of primitive concepts that belong to the target class or the contrasting class(es). For example, the tuple whose value in attribute "Category" is in {M.S., M.A., Ph.D} should be retrieved for the class "Graduate", and the tuple whose value in attribute "Category" is in {freshman, sophomore, junior, senior} should be retrieved for the class "undergraduate".

A set of conceptual biases is coded in a conceptual bias file which is organized as a three-column table. The first column contains the lower level concepts, the second column contains the corresponding higher level concepts, and the third column specifies the attribute names for which the conceptual bias serves. Such organization facilitates the search of the conceptual bias during the learning process. Table 7.1 is a sample table which stores the conceptual bias for the attribute "Category".

freshman	undergraduate	Category
sophomore	undergraduate	Category
junior	undergraduate	Category
senior	undergraduate	Category
M.S.	graduate	Category
M.A.	graduate	Category
Ph.D.	graduate	Category

Table 7.1. The conceptual bias table for attribute "Category".

The learning program consists of two modules which implement the LCHR algorithm and the LCLR algorithm, respectively. Either of these two modules can be invoked by user's command. The entire learning process can be monitored if a

specific parameter, *watch*, is set on. That is, every intermediate generalized relation will be printed on the terminal.

After the final generalized relation is derived, the DBLEARN system transforms the learning result to the corresponding logic formula and reports to the user through the user-interface.

7.2. Experimental Results

We have performed several experiments on different data domains to test the LCHR algorithm and the LCLR algorithm. We report the results of two experiments here.

Experiment 7.1. Learning a characteristic rule and a classification rule for graduate students.

In this experiment, we test the DBLEARN system using the same data presented in Table 3.1 and the same conceptual bias depicted in Figure 3.1 and Figure 3.2.

Learning-Request 7.1:

learn characteristic_rule for graduate
from relation student
relevant to attributes Category, Major, Birth_Place, GPA
with threshold = 3

The DBLEARN system generates the following output.

The final generalized relation

Major	Birth_Place	GPA
art	Canada	excellent
science	Canada	excellent
science	foreign	good

The characteristic rule for graduate students is:

graduate(x) =>

((Birth_Place(x) = Canada) and (GPA(x) = excellent))

or

((Major(x) = science) and (Birth_Place(x) = foreign) and (GPA(x) = good))

Learning-Request 7.2:

learn classification_rule for graduate
from relation student
relevant to attributes Category, Major, Birth_Place, GPA
with threshold = 3

The following is the output generated by the DBLEARN system.

The final generalized relation

Major	Birth_Place	GPA
science	foreign	good

The classification rule for graduate students is:

graduate(x) <=

(Major(x) = science) and (Birth_Place(x) = foreign) and (GPA(x) = good)

Obviously, the system performs the learning process correctly. The learning results are the exact ones that we expect.

Experiment 7.2. Learning characteristic rules and classification rules from a bank relation.

Suppose Table 7.2 is a relation of a bank database with attributes Name, Age, Address, Occupation and Balance. This experiment is designed to learn some characteristic rules and classification rules from this relation.

Name	Age	Address	Occupation	Balance
Addison	58	W_Van	Professor	40301
Aiken	34	N_Van	Engineer	34111
Ajtai	31	Burnaby	Engineer	26449
Allen	27	Coquitlam	Instructor	16033
Andrews	26	Delta	Instructor	15678
Baker	38	Burnaby	Professor	32661
Bent	55	Van	Doctor	40442
Booth	47	Surrey	Farmer	10309
Bray	40	W_Van	Engineer	25376
Brown	36	Delta	Plumber	11076
Chew	24	Coquitlam	Mechanic	6733
Chin	48	W_Van	Doctor	50644
Church	42	Van	Professor	25111
Clarke	52	Burnaby	Mechanic	14659
Edmonds	30	Surrey	Mechanic	10525
Elias	44	N_Van	Professor	29883
Fateman	27	Delta	Farmer	4879
Fischer	35	Burnaby	Engineer	24965
Fox	50	Surrey	Farmer	6022
Funt	34	N_Van	Doctor	60933
Gale	23	Burnaby	Instructor	29972
Gilbert	38	N_Van	Professor	25001
Gray	57	Coquitlam	Instructor	10281
Hall	29	Coquitlam	Plumber	25828
Johnson	25	Surrey	Mechanic	5624

(to be continued)

(continued)

Name	Age	Address	Occupation	Balance
Kim	37	W_Van	Engineer	30566
Lam	22	Surrey	Farmer	5488
Lee	63	Van	Doctor	60028
Levin	33	Surrey	Mechanic	10977
Miller	45	W_Van	Engineer	25446
Meyer	31	N_Van	Professor	33892
Nelson	56	Delta	Farmer	5022
Nerode	27	Burnaby	Mechanic	30645
Orlin	33	W_Van	Engineer	39487
Park	28	Delta	Farmer	9080
Partson	43	Burnaby	Professor	31625
Pinter	51	Van	Engineer	25556
Rabin	33	Coquitlam	Mechanic	24667
Rackoff	55	Burnaby	Doctor	30023
Reif	57	Burnaby	Mechanic	21085
Rem	38	N_Van	Professor	35574
Rogers	38	Burnaby	Mechanic	15633
Rosser	35	Delta	Mechanic	10884
Sacks	44	W_Van	Engineer	41724
Shearer	33	Surrey	Farmer	16787
Simon	46	Van	Professor	33345
Smith	44	Delta	Farmer	15000
Snir	25	Coquitlam	Engineer	22044
Swamy	27	N_Van	Engineer	32011
Tenney	47	Van	Professor	40463
Tompa	37	Surrey	Mechanic	10552
Tsou	52	Delta	Farmer	5399
Tucker	47	N_Van	Professor	45274
Valiant	36	W_Van	Engineer	45101
Vitter	29	Surrey	Mechanic	12409
Wood	21	Surrey	Farmer	5899
Wyllie	55	W_Van	Doctor	60322
Yao	55	Van	Professor	24773
Young	42	Burnaby	Engineer	30448
Yun	62	N_Van	Professor	46225
Zaks	29	Delta	Plumber	5987

Table 7.2. A sample relation *Bank*.

Suppose Figure 7.3 is the specified set of concept hierarchies.

{ 21 — 30 } ⊂ 20-30
{ 31 — 40 } ⊂ 30-40
{ 41 — 50 } ⊂ 40-50
{ 51 — 60 } ⊂ 50-60
{ 61 — 70 } ⊂ 60-70
{ 20-30 } ⊂ young_age
{ 30-40, 40-50 } ⊂ mid_age
{ 50-60, 60-70 } ⊂ old_age
{ W_Van, Van } ⊂ West_part
{ Burnaby, Coquitlam } ⊂ East_part
{ N_Van } ⊂ North_part
{ Surrey, Delta } ⊂ South_part
{ Professor, Doctor, Engineer } ⊂ professional
{ Instructor, Mechanic, Farmmer, Plumber } ⊂ non_professional
{ 5k — 10k } ⊂ low
{ 11k — 20k } ⊂ average
{ 21k — 35k } ⊂ mid_high
{ 36k — 60k } ⊂ high

Figure 7.3. A concept hierarchy of the bank database

As defined in Figure 7.3, *professional* occupations include *Professor, Doctor* and *Engineer*. And the *non_professional* ones include *Instructor, Technician, Farmmer* and *Plumber*. The following query invokes the DBLEARN system to discover the characteristic rule for professionals.

Learning-Request 7.3:

learn characteristic_rule for professional
from relation bank
relevant to attributes Age, Address, Occupation, Balance
with threshold = 5

The DBLEARN system generates the following output.

The final generalized relation

Age	Address	Balance
ANY	West_part	high_bal
ANY	West_part	mid_high_bal
ANY	North_part	high_bal
ANY	East_part	mid_high_bal
ANY	North_part	mid_high_bal

The characteristic rule for professional people is:

professional(x) =>
((Address(x) = West_part) and (Balance(x) = [high_bal , mid_high_bal]))
or
((Address(x) = North_part) and (Balance(x) = [high_bal , mid_high_bal]))
or
((Address(x) = East_part) and (Balance(x) = mid_high_bal))

If we want to learn a knowledge rule which distinguishes the professional people from the non_professional people, Learning-Request 7.4 can be posted to DBLEARN system.

Learning-Request 7.4:

learn classification_rule for professional *vs* non_professional
from relation bank
relevant to attributes Age, Address, Occupation, Balance
with threshold = 5

The DBLEARN system generates the following learning result.

The final generalized relation

Age	Address	Balance
ANY	West_part	high_bal
ANY	West_part	mid_high_bal
ANY	North_part	high_bal
ANY	North_part	mid_high_bal

The classification rule for professional people versus non_professional people is:

professional(x) <=
((Address(x) = West_part) and (Balance(x) = [high_bal , mid_high_bal]))
or
((Address(x) = North_part) and (Balance(x) = [high_bal , mid_high_bal]))

Notes that this rule is slightly different from the rule derived from Learning-Request 7.3. The tuple

((Address(x) = East_part) and (Balance(x) = mid_high_bal))

appears in the characteristic rule for professionals, but not in the classification rule for professionals. The obvious reason is that among the people who are living in the east_part of Vancouver area and have a mid-high balance in the bank, some are professionals, but some are not. Thus, it is not possible to determine whether a person *x* is a professional, based on "Address(x)" and "Balance(x)". The DBLEARN system sets marks on these kinds of tuples in the generalized relation and removes them from the representation of classification rules. This process enable the classification rules to discriminate the concepts in the target class from the contrasting classes.

CHAPTER 8

CONCLUSIONS AND FUTURE RESEARCH

8.1. Conclusions

It is attractive and challenging to automatically compute generalization rules from large databases. In this thesis, we have studied the methods of learning characteristic rules and classification rules from relational databases and have developed two efficient database learning algorithms, **LCHR** and **LCLR**. These algorithms adopt the attribute-oriented induction approach, integrate database operations with the learning processes, and provide an efficient way of extracting knowledge from databases.

The LCHR algorithm is designed for learning characteristic rules from relational databases. It adopts an attribute-oriented concept tree ascending technique which substitutes the lower-level concepts of the attribute in a tuple by its corresponding higher-level concepts and thus generalizes the relation. By eliminating the redundant tuples and applying a threshold value to control the generalization process, the final generalized relation consists of only a small number of tuples which can be transformed to a simple logic formula.

The LCLR algorithm is designed for learning classification rules from relational databases. Similar to the LCHR algorithm, the LCLR algorithm also applies the attribute-oriented concept tree ascending technique. However, this algorithm detects the overlapping tuples in the learning process and removes such tuples from the final

generalized relation, thus ensuring that the learned concepts can be distinguished from the concepts in other classes.

In order to cope with different learning situations, some variations of these learning algorithms have been studied. By developing various kinds of techniques, our approach can handle different types of concept hierarchies. By adjusting thresholds in the learning process, our approach can learn knowledge rules in different conjunctive and disjunctive forms. By incorporating the statistical techniques, our learning algorithms can discover qualitative rules with quantitative information and handle noisy data and exceptional cases elegantly.

A comparison of our approach with many other algorithms for learning from examples shows that our algorithms have many distinct features, such as, the ability to use database facilities, learn disjunctive rules, handle overlapping instances, provide quantitative information, and handle noisy data and exceptional cases. Our analysis of the algorithms demonstrates that the attribute-oriented induction approach substantially reduces the complexity of the database learning process.

8.2. Future Research

There are many interesting research issues related to learning from large databases.

8.2.1. Applications of Knowledge Rules Discovered from Relational Databases

Our learning system can learn characteristic rules and classification rules from relational databases effectively. An immediate issue is the application of knowledge

rules discovered in the learning process.

The knowledge rules learned from relational database are very useful in many applications, some of which are listed below:

(1) *Discovery of knowledge rules for knowledge-base systems and expert systems.*

Since rules are derived from a huge number of data stored in a relational database, they represent important knowledge about data in the database. Thus, our approach is an important method to obtain knowledge rules for knowledge-base systems and expert systems.

(2) *Processing of queries which involve abstract concepts.*

In general, relational databases can only answer queries which involve the concepts presented in the database, but they cannot handle queries like "What are the major characteristics of a graduate student?" and "How can we describe the major differences between graduate students and undergraduate students?". Such queries involve concepts which are at a higher level than the primitive data stored in relational databases. By applying the knowledge rules obtained by our learning algorithms, it is possible to answer such learning-requests in a natural way.

(3) *Semantic query optimization using the learned rules.*

Some queries can be answered more efficiently by the learned knowledge rules without searching databases [CGM88]. For example, the query, "Is there any foreign student in the undergraduate program?", usually indicates that the relation *undergraduate student* must be searched. However, if the characteristic

rule indicates that there are no undergraduate students who come from other countries, this query can be answered immediately without any search. A similar situation occurs when the query is "Are all graduate students studying science?". If the characteristic rule shows that the major of some students is "science", but the major of some other students is "art", then the obvious answer to this query is "No". Clearly, learned rules may speed up or optimize the database query processing as previously studied in semantic query optimization [CGM88]. Notice that when there is a large number of learned rules, it is nontrivial to search such a rule space. In such a case, there is a trade-off between performing such semantic optimization versus searching database directly. More detailed study in semantic query optimization using generalized rules may produce some interesting results.

8.2.2. Construction of an Interactive Learning System

As shown in our learning system, the database learning process is guided by experts or users. Experts and users must provide the conceptual bias, specify the learning task and define the threshold value. It is important to obtain such information by interaction with users and experts. We propose to build an interactive learning system which should provide the following features.

(1) The system should have a user-friendly interface to facilitate users' communication with the learning system. A more flexible database learning language should be developed for such an interface.

(2) The entire learning process should be monitored and controlled by users. For example, at some stage of the learning process, users may terminate the generalization on some selected attributes but continue the process on other attributes. In order to obtain multiple rules, users may influence the learning process using different threshold values.

8.2.3. Discovery of Concept Hierarchies

The current system DBLEARN needs users or domain experts to provide the conceptual bias explicitly. We have examined in Chapter 7 some techniques for automatic generation of concept hierarchies. Further research on this topic, especially on the discovery of concept hierarchies for discrete attributes, should be studied in depth. The following possible approaches should be considered in future research.

(1) Develop some specific mapping techniques which can map the discrete attribute values to numeric values, and then apply cluster analysis and numeric taxonomy to discover the concept hierarchies [ChF85, Fis87].

(2) Design an efficient method to search the concept hierarchies stored in the relational database, either from data or from integrity constraints.

8.2.4. Performance Testing

To further study the efficiency of the database learning algorithms, we plan to perform a systematic study and some performance testing on different learning algorithms in databases. Moreover, the complexity measurement of our algorithms will

be analyzed in more detail. Further improvement of our techniques will be examined and experimented. The study will finally lead to a comprehensive efficient database learning system.

REFERENCES

References

[BuM78] B. G. Buchanan and T. M. Mitchell, Model-Directed Learning of Production Rules, *in Waterman et. al. (eds.), Pattern-Directed Inference System, Academic Press, 1978, 297-312.*

[CCH89a] Y. Cai, N. Cercone and J. Han, Learning Characteristic Rules from Relational Databases, *Proceedings of the International Symposium Computational Intelligence'89*, Milano, Italy, September 1989.

[CCH89b] Y. Cai, N. Cercone and J. Han, Attribute-Oriented Induction in Relational Databases, *Proceedings of IJCAI-89 Workshop on Knowledge Discovery in Databases*, Detroit, Michigan, August 1989, 26–36.

[CCH90] Y. Cai, N. Cercone and J. Han, An Attribute-Oriented Approach for Learning Classification Rules from Relational Databases, *Proceedings of the 6th International Conference on Data Engineering*, Los Angeles, CA, February 1990.

[CGM88] U. S. Chakravarthy, J. Grant and J. Minker, Foundations of Semantic Query Optimization for Deductive Databases, *In J. Minker (ed.), Foundations of Deductive Databases and Logic Programming*, Morgan Kaufmann, 1988, 243–274.

[ChF85] Y. Cheng and K. S. Fu, Conceptual Clustering in Knowledge Organization, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 7, No.5, September 1985, 592–598.

[CoF83] P. Cohen and E. A. Feigenbaum, *The Handbook of Artificial Intelligence (Vol. III)*, Heuristic Press & William Kaufmann Inc., 1983.

[DiM81] T. G. Dietterich and R. S. Michalski, Inductive Learning of Structural Descriptions: Evaluation Criteria and Comparative Review of Selected Methods, *Artificial Intelligence* , Vol. 16, 1981, 257–294.

[DiM83] T. G. Dietterich and R. S. Michalski, A Comparative Review of Selected Methods for Learning from Examples, *in Michalski et. al. (eds.), Machine Learning: An Artificial Intelligence Approach, Vol. 1, Morgan Kaufmann, 1983, 41–82.*

[FaM86] B. C. Falkenhainer and R. S. Michalski, Integrating Quantitative and Qualitative Discovery: the ABACUS system, *Machine Learning* , Vol. 1, No. 4 , 1986, 367-401.

[Fis87] D. Fisher, Improving Inference Through Conceptual Clustering, *Proceedings of 1987 AAAI Conference*, Seattle, Washington, July 1987, 461–465.

[Fis88] D. Fisher, A Computational Account of Basic Level and Typicality Effects, *Proceedings of 1988 AAAI Conference* , Saint Paul, Minnesota, August 1988, 233–238.

[GMN84] H. Gallaire, J. Minker and J. Nicolas, Logic and Databases : A Deductive Approach, *ACM Computing Surveys* , 16(2), 1984, 153–185.

[GeN87] M. Genesereth and N. Nilsson, *Logical Foundations of Artificial Intelligence*, Morgan Kaufmann, 1987.

[Hau86] D. Haussler, Quantifying the Inductive Bias in Concept Learning, *Proceedings of 1986 AAAI Conference* , Philadelphia, PA, August 1986, 485–489.

[Hau87a] D. Haussler, Learning Conjuctive Concepts in Structural Domains, *Proceedings of 1987 AAAI Conference*, Seattle, Washington, July 1987, 466–470.

[Hau87b] D. Haussler, Bias, Version Spaces and Valiant's Learning Framework, *Proceedings of the 4th International Workshop on Machine Learning*, Irvine, CA, 1987, 324–336.

[HaM77] F. Hayes-Roth and J. McDermott, Knowledge Acquisition from Structural Descriptions, *Proceedings of 5th International Joint Conference on Artificial Intelligence* , Cambridge, MA, August 1977, 356–362 .

[HaM78] F. Hayes-Roth and J. McDermott, An Interference Matching Technique for Inducing Abstractions, *Communications of the ACM*, Vol. 21, No.5 , 1978, 401-410.

[ImC83] R. L. Iman and W. J. Conover, A Modern Approach to Statistics, , 1983.

[KMK89] K. A. Kaufman, R. S. Michalski and L. Kerschberg, Mining for Knowledge in Databases: Goals and General Description of the INLEN System, *Proceedings of IJCAI-89 Workshop on Knowledge Discovery in Databases*, Detroit, Michigan, August 1989, 158–172.

[KKM88] D. G. Kleinbaum, L. L. Kupper and K. E. Muller, Applied Regression Analysis and Other Multivariable Methods, , 1988.

[Kok86] M. M. Kokar, Coper: a Methodology for Learning Invariant Functional Descriptions, *in Michalski et. al. (eds.), Machine Learning: a Guide to Current Research, Cluwer Academic Publishers, 1986, 151–154.*

[KoS86] H. F. Korth and A. Silberschatz, *Database System Concepts*, McGraw-Hill, 1986.

[KuS88] D. Kulkarni and H. A. Simon, The Process of Scientific Discovery: The Strategy of Experimentation, *Cognitive Science*, Vol. 12, 1988, 139-175.

[Lan77] P. W. Langley, Rediscovering Phisics with BACON.3, *Proceedings of the 5th IJCAI Conference*, Cambridge, MA, 1977, 505–507.

[LBS83] P. W. Langley, G. L. Bradshaw and H. A. Simon, Rediscovering Chemistry with the BACON System, *in Michalski et. al. (eds.), Machine Learning: An Artificial Intelligence Approach, Vol. 1, Morgan Kaufmann, 1983, 307–330.*

[Len77] D. B. Lenat, On Automated Scientific Theory Formation: a Case Study Using the AM Program, *In J. E. Hayes, D. Michie, and L. I. Mikulich (eds.), Machine Intelligence 9*, Halsted Press, 1977, 251–286.

[Lub89] D. J. Lubinsky, Discovery from Database: A Review of AI and Statistical Techniques, *Proceedings of IJCAI-89 Workshop on Knowledge Discovery in Databases*, Detroit, Michigan, August 1989, 204–218.

[MaK87] M. V. Manago and Y. Kodratoff, Noise and Knowledge Acquisition, *Proceedings of the 10th IJCAI Conference*, Milan, Italy, 1987, 348–354.

[MiC80] R. S. Michalski and R. L. Chilausky, Learning by Being Told and Learning from Examples: An Experimental Comparison of the Two methods of Knowledge Acquisition in the Context of Developing an Expert System for Soybean Disease Diagnosis , *International Journal of Policy Analysis and Information System* , Vol. 4, 1980, 125–161.

[Mic83] R. S. Michalski, A Theory and Methodology of Inductive Learning, *in Michalski et. al. (eds.), Machine Learning: An Artificial Intelligence Approach, Vol. 1, Morgan Kaufmann, 1983, 83–134.*

[MiS83] R. S. Michalski and R. Stepp, Automated Construction of Classifications: Conceptual Clustering Versus Numerical Taxonomy, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 5, 1983, 396–410.

[MCM83] R. S. Michalski, J. G. Carbonell and T. M. Mitchell, *Machine Learning, An Artificial Intelligence Approach, Vol. 1*, Morgan Kaufmann, 1983.

[MMH86] R. S. Michalski, L. Mozetic, J. Hong and N. Lavrac, The Multi-purpose Incremental Learning System AQ15 and Its Testing Application to Three Medical Domains, *Proceedings of 1986 AAAI Conference* , Philadelphia, PA, 1986, 1041–1045.

[MCM86] R. S. Michalski, J. G. Carbonell and T. M. Mitchell, *Machine Learning, An Artificial Intelligence Approach, Vol. 2*, Morgan Kaufmann, 1986.

[Mic87] R. S. Michalski, How to Learn Imprecise Concepts: A Method for Employing a Two-tiered Knowledge Representation in Learning, *Proceedings of the 4th International Workshop on Machine Learning*, Irvine, CA, 1987, 50–57.

[Mit77] T. M. Mitchell, Version Spaces: A Candidate Elimination Approach to Rule Learning, *Proceedings of the 5th IJCAI Conference*, Cambridge, MA, 1977, 305–310.

[Mit79] T. M. Mitchell, An Analysis of Generalization as a Search Problem, *Proceedings of the 6th IJCAI conference*, Tokyo, Japan, 1979, 577–582.

[Mit82] T. M. Mitchell, Generalization as Search, *Artificial Intelligence*, Vol. 18, 1982, 203-226.

[Pia89] G. Piatetsky-Shapiro, Discovery of Strong Rules in Databases, *Proceedings of IJCAI-89 Workshop on Knowledge Discovery in Databases*, Detroit, Michigan, USA, August 1989, 264–274.

[Qui83] J. R. Quinlan, Learning Efficient Classification Procedures and Their Application to Chess End-Games, *in Michalski et. al. (eds.), Machine Learning: An Artificial Intelligence Approach, Vol. 1, Morgan Kaufmann, 1983, 463–482.*

[Qui86] J. R. Quinlan, The Effect of Noise on Concept Learning, *in Michalski et. al. (eds.), Machine Learning: An Artificial Intelligence Approach, Vol. 2,*

Morgan Kaufmann, 1986, 149–166.

[Rei84] R. Reiter, Towards a Logical Reconstruction of Relational Database Theory, *in M. Brodie, J. Mylopoulos, and J. Schmidt (Eds.), On Conceptual Modeling, Spring-Verlag, 1984, 191–233.*

[Ren86] L. Rendell, A General Framework for Induction and a Study of Selective Induction, *Machine Learning*, 1, 1986.

[Rus88] S. J. Russell, Tree-Structured Bias, *Proceedings of 1988 AAAI Conference*, Minneapolis, Minnesota, August 1988, 641–645.

[Ste87] R. E. Stepp, Concepts in Conceptual Clustering, *Proceedings of the 10th IJCAI Conference*, Milan, Italy, August 1987, 211–213.

[SuF86] D. Subramanian and J. Feigenbaum, Factorization in Experiment Generation, *Proceedings of 1986 AAAI Conference*, Philadelphia, Pennsylvania, August 1986, 518–522.

[Ull89] J. D. Ullman, *Principles of Database and Knowledge-Base Systems, Vols. 1 & 2*, Computer Science Press, 1989.

[Ver75] S. A. Vere, Induction of Concepts in the Predicate Calculus, *Proceedings of the 4th International Joint Conference on Artificial Intelligence*, Los Altos, CA, 1975, 281-287.

[WaE87] L. Watanabe and R. Elio, Guiding Constructive Induction for Incermental Learning from Examples, *Proceedings of the 10th IJCAI Conference*, Milan, Italy, August 1987, 293–296.

[WGT87] S. M. Weiss, R. S. Galen and P. V. Tadepalli, Optimizing the Predictive Value of Diagnostic Decision Rules, *Proceedings of 1987 AAAI conference*, Seattle, Washington, July 1987, 521–526.

[Win75] P. Winston, Learning Structural Descriptions from Examples, *in Winston, P. (eds.), The Psychology of Computer Vision, McGraw-Hill, 1975, 157-209.*

[WoC88] A. K. C. Wong and K. C. C. Chan, Learning from Examples in the Presence of Uncertainty, *Proceedings of International Computer Science Conference'88*, Hong Kong, December 1988, 369–376.

[Zyt87] J. M. Zytkow, Combining Many Searches in the FAHRENHEIT Discovery System, *Proceedings of the 4th International Workshop on Machine Learning*, Irvine, CA, 1987, 281–287.