Data-Driven Discovery of Quantitative Rules in Relational Databases

Jiawei Han, Yandong Cai, and Nick Cercone, *Member, IEEE*

Abstract—**A quantitative rule is a rule associated with quantitative information which assesses the representativeness of the rule in the database. In this paper, an efficient induction method is developed for learning quantitative rules in relational databases. With the assistance of knowledge about concept hierarchies, data relevance, and expected rule forms, attribute-oriented induction can be performed on the database, which integrates database operations with the learning process and provides a simple, efficient way of learning quantitative rules from large databases. Our method learns both characteristic rules and classification rules. Quantitative information facilitates quantitative reasoning, incremental learning, and learning in the presence of noise. Moreover, learning qualitative rules can be treated as a special case of learning quantitative rules. Our paper shows that attribute-oriented induction provides an efficient and effective mechanism for learning various kinds of knowledge rules from relational databases.**

Index Terms— **Knowledge discovery in databases, machine learning, attribute-oriented induction, quantitative rules, characteristic rules, classification rules, data-driven learning algorithms.**

I. Introduction

THE growth in the number of available databases far outstrips the growth of corresponding knowledge bases. This creates both a need and an opportunity for extracting knowledge from databases. By learning from databases, interesting relationships among data can be discovered automatically, and the extracted knowledge may facilitate deductive reasoning and query processing in database systems.

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 a conjunctive normal form. Such uniformity, together with the well-developed relational technology [19], [21], facilitate the development of efficient database learning algorithms.

Different kinds of knowledge rules can be learned from databases. A learned rule can be either a *qualitative rule* or a *quantitative rule*, where the former does not associate quantitative information whereas the latter does. For example, the statement, *the salaries of professors of Applied Sciences are high*, is a qualitative rule while the statement, *the salaries of 60% professors of Arts are high*, is a quantitative rule. A quantitative rule provides statistical information about the rule, which facilitates quantitative reasoning, incremental learning, and learning in the presence of noise and exceptions.

From another point of view, a learned rule can be either a *characteristic rule* or a *classification rule*. A **characteristic rule** is an assertion which characterizes the concept satisfied by all of the relevant data in the database. For example, the symptoms of a particular disease can be summarized as a characteristic rule. Alternatively, a **classification rule** is an assertion which discriminates the concepts of one class from others. For example, to distinguish one disease from others, a classification rule summarizes the symptoms that discriminate this disease from others.

A major challenge of learning in databases is computational efficiency. Our approach strives for efficiency in two aspects: i) knowledge-directed learning and ii) attribute-oriented induction. The former is achieved by providing knowledge about the learning task, the concept hierarchies, and the expected rule forms. The latter is achieved by attribute-oriented concept tree ascension. These techniques substantially reduce the search space and improve the efficiency in a database learning process.

In this paper, our study is on the learning of quantitative rules from relational databases. The paper is organized as follows. The primitives required for learning from databases are discussed in Section II. Learning principles and algorithms are presented in Section III. Variations of these learning algorithms are discussed in Section IV. A comparison of our method with others is presented in Section V. The application of learned rules is discussed in Section VI, and the concluding remarks are provided in Section VII.

II. Knowledge Required for Learning from Databases

Given a number of facts, generalization can be performed in many different directions [6], [13]. In order to constrain a generalization process and extract interesting rules from databases, learning should be directed by background knowledge, such as knowledge contained in *concept hierarchies*, and learning task specifications, such as the specifications of task-relevant data (*data relevance*) and expected rule forms (*rule expectance*).

A. Concept Hierarchy

The concept hierarchy provides valuable information for inductive learning [9], [18]. By organizing different levels of concepts into a taxonomy, candidate rules can be restricted to formulas with a particular vocabulary (*conceptual bias* [9])

Manuscript received December 19, 1989; revised September 18, 1990. This work was supported in part by the Natural Sciences and Research Council of Canada under Operating Grants A-3723 and A-4309 and by a research grant from the Centre for Systems Science, Simon Fraser University.

The authors are with the School of Computing Science, Simon Fraser University, Burnaby, B.C., Canada V5A 1S6.

IEEE Log Number 9205833.

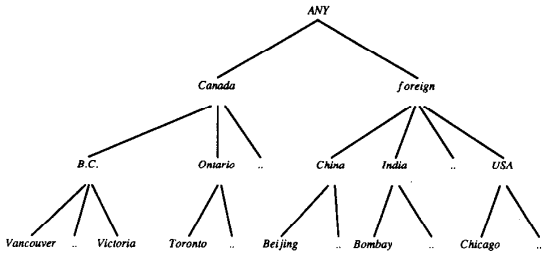


Fig. 1. A concept hierarchy for the attribute *City*.

and be described by higher level concepts, which permits a learned rule to be represented in a simple and explicit form.

Knowledge about concept hierarchies can be directly provided by domain experts. A concept hierarchy is often related to a specific attribute and is partially ordered according to general-to-specific ordering. The most general point is the null description (*ANY*), and the most specific points correspond to the specific values of an attribute in the database. For example, the concept hierarchy of an attribute *City* may form a taxonomy as shown in Fig. 1. Although a database could be large, a concept hierarchy can be organized by experts with a reasonable effort because only *distinct* attribute values may correspond to the nodes in a concept hierarchy of that attribute.

Concept hierarchy information may also be implicitly stored in the database. For example, the information about Fig. 1 may actually be stored in a data relation. In order to utilize the information, the hierarchical relationship among attributes should be explicitly indicated at the schema level. For example, the hierarchical relationship for the attribute *City* can be indicated in the database schema as "*city* **in** *province* **in** *country*," (that is, *city* is a lower level concept of its *province*, etc.). Then the taxonomy about the attribute *City* can be retrieved from the database and be used in the learning process.

Alternatively, a concept hierarchy can be constructed automatically based on clustering behavior and database statistics. The automatic construction of discrete attributes from numerical values can be performed by first obtaining the distribution of attribute values in the database and then setting the ranges of the values and performing more detailed classifications for more densely clustered ranges. For example, the concept hierarchy for an attribute grade point average (GPA) of graduate students can be constructed based on the database statistics. Suppose that all the GPA's are between 0–4, and most GPA's for graduates are clustered between 3 and 4. Then we may classify the values into three clases, [0–1.99], [2–2.99], and [3–4], and then perform a more detailed classification for those in [3–4]. Such a statistical technique can also be performed on the attributes with discrete values under certain circumstances [7]. For example, if the birth place of most employees are clustered in Canada and scattered in many different countries, the top-most level (except NULL) concepts for birth place can be categorized as *Canada* and *foreign*.

In our discussion, we assume that the concept hierarchies are given and are in the form of balanced concept trees. The handling of other kinds of concept hierarchies is discussed in Section IV.

B. Database Relevance

Although a relational database stores a large amount of data, usually only a portion of it is relevant to a specific learning task. Clearly, *preprocessing* should be performed to extract and group the task relevant data from a database before generalization. The preprocessing can be viewed as a relational query which takes a learning request as a retrieval command to search for the necessary sets of data from the database and group them according to the learning task.

When learning a characteristic rule, the set of data being considered (undergoing learning) is called the **target class** of the learning process. When learning a classification rule, it is necessary to collect the set of data being considered (undergoing learning) and the set(s) of data being used for contrasting. In this case, we call the set of data being considered the **target class** and the set(s) of data used for contrasting the **contrasting class(es).** For example, to extract the characteristics of professors in Computing Science, only the data relevant to those professors are retrieved and grouped into one class, the target class. To distinguish the professors in Computing Science from the instructors in the same department, only the data relevant to them are retrieved but are grouped into two classes, the target class for those about professors and the contrasting class for those about instructors.

In most **learning-from-examples** algorithms [6], [9], the examples undergoing learning are partitioned into positive and negative sets. However, since a relational database does not store negative data in general, there are usually no explicitly specified negative examples. When learning a characteristic rule, we should bear in mind that there are no negative examples for specialization, and the generalization on the data in the target class should be performed cautiously to avoid over-generalization. Alternatively, when learning a classification rule which distinguishes the properties in the target class from those in the contrasting class(es), we may treat the tuples in the target class as positive data and those in the contrasting class as "negative" data. However, we should notice that the data in the contrasting class(es) do not imply that similar data cannot appear in the target class, but imply that they cannot be used to distinguish the target class from the contrasting class. Thus such a kind of data is different from the negative data in the target class. Therefore, generalization should still be performed conservatively to avoid over-generalization. Data in the contrasting class are used to exclude the properties shared by both classes. Such an exclusion of the shared data results in a condition which is sufficient but may not be necessary for the data to be in the target class. This will be explained further in Section III.

Preprocessing can be specified by a relational-like language interface and implemented by relational operations. For example, the join and projection operations are often necessary to collect data from *several* task relevant relations. The *group by* operation (in SQL syntax) is useful at clustering data according to the target class and the contrasting classes, respectively, for learning classification rules. Even aggregate operations could be useful if the task is relevant to some aggregate properties.

TABLE I
A TUPLE IS ESSENTIALLY A LOGICAL FORMULA

Name	Sex	Age	Birth_place	Department	Position	Salary
Benson	male	45	Vancouver	cmpt	full_prof	63 000

A learning task may refer to some *nonprimitive data*, where **primitive data** are those data stored in data relations and **nonprimitive data** are those data appearing only in concept hierarchies. In this case, nonprimitive data should be mapped to the primitive ones in order to find the task relevant data in the database. For example, suppose the concept hierarchy contains the information about the professors such as "{ *assist_prof*, *assoc_prof*, *full_prof*} $\subset$ *professor*" and the database contains a relation *Employee* with the scheme:

Employee = (*Name*, *Sex*, *Age*, *Birth_place*, *Department*, *Position*, *Salary*).

If the task is to learn a characteristic rule which describes the relationship among *position*, *age*, and *salary* of the professors in Computing Science, the corresponding SQL query for preprocessing should be

select *Position*, *Age*, *Salary*
from *Employee*
where *Position* = "*professor*" **and** *Department* = "*Computing Science*."

Notice that "*professor*" is *nonprimitive* data, which should be mapped to the set of corresponding primitive data in query processing, that is, the SQL expression

Position = "*professor*"

should be mapped to

Position $\in$ {"*assist_prof*," "*assoc_prof*," "*full_prof*"},

or, in SQL:

(*Position* = "*assist_prof*" **or** *Position* = "*assoc_prof*" **or** *Position* = "*full_prof*").

By preprocessing, data relevant to the learning task are collected into one relation, which is called the **initial data relation**. Our later discussion is focused on the induction process on the initial data relation.

C. Rule Expectance

According to the theory of logic and databases [8], [21], data in relational databases correspond to logical formulas in the first-order logic. Each tuple in a relation can be viewed as a logical formula in conjunctive normal form. For example, the tuple in Table I represents a logical formula (1). Multiple tuples can be viewed as a logical formula in disjunctive normal form.

1) $\exists x$ ((*Name*(x) = Benson) $\wedge$ (*Sex*(x) = male) $\wedge$ (*Age*(x) = 45) $\wedge$ (*Birth_place*(x) = Vancouver) $\wedge$ *Department*(x) = cmpt $\wedge$ (*Position*(x) = full_prof) $\wedge$ (*Salary*(x) = 63 000)).

Similarly, the intermediate and final learning results can be represented in relational forms called **generalized relations.** Learning can be viewed as a sequence of processes each of which transforms a less generalized relation to a more generalized relation. Each generalized relation is essentially a logical formula in disjunctive normal form or a conjunctive normal form if it contains only one tuple.

To control the learning process, it is often necessary to specify the expected rule form by a **generalization threshold,** which is the maximum number of tupels in the target class of the final generalized relation, that is, the maximum number of disjuncts in the resulting formula. The generalization threshold should be a small positive integer which indicates the expected maximum number of tuples in the final generalized relation. The threshold value can be specified or predefined by a user, a database administrator, or a domain expert. There is a trade-off between a small threshold value and a moderately large one. A small threshold leads to a simple rule with a few disjuncts. However, it may result in over-generalization and the loss of some valuable information. On the other hand, a relatively large threshold may preserve some useful information, but it may result in a relatively complex rule with many disjuncts and some semi-generalized results. Therefore, fine tuning of thresholds is often necessary in the learning process; this fine tuning can be performed interactively by domain experts and/or users.

If the threshold value is one, the final generalized relation contains only one tuple (*a conjunctive rule*). Otherwise, it contains a small number of tuples (*a disjunctive rule*). Although many learning algorithms [13] can learn only conjuctive rules, it is necessary to provide the flexibility to learn conjunctive rules as well as disjunctive rules in database applications because of the diversity of data in large databases.

III. PRINCIPLES AND ALGORITHMS FOR LEARNING FROM DATABASES

We first present the principles and algorithms for learning characteristic rules from databases and then extend them to learning classification rules.

A. Learning Characteristic Rules

We examine the learning of characteristic rules in databases using Example 1.

Example 1: Table II depicts a portion of a data relation *Employee* in a university database. Suppose our task is to *learn a quantitative characteristic rule for professors in Applied Sciences relevant to attributes Name, Sex, Age, Birth-place, and Salary.*

Suppose further that the concept hierarchy table is as shown in Fig. 2, where $A \subset B$ indicates that B is a generalized concept of A, and ANY(attribute), such as ANY(*Position*), represents the most general concept for the attribute, such as *Position*. Each specified concept hierarchy represents a taxonomy of concepts in an attribute domain (a concept tree like Fig. 1).

Before beginning the induction process, preprocessing is performed which collects the data relevant to the learning task. Notice that *professors* and *Applied Sciences* are nonprimitive data which should be mapped to the primitive data using the concept hierarchy table.

We now turn our attention to the generalization process. First, possible generalizations should be considered on each attribute rather than on composite attributes. We then have the following strategy:

TABLE II
A RELATION *Employee* IN A UNIVERSITY DATABASE

Name	Sex	Age	Birth_place	Department	Position	Salary
Anderson	female	26	Burnaby	cmpt	secretary	26 000
Bach	male	38	Ottawa	electr_eng	lab_manager	41 000
Barton	female	30	Toronto	chem	junior_lecturer	28 000
Benson	male	45	Vancouver	cmpt	full_prof	63 000
...	...	...	...	...	...	...
Winton	male	38	Seattle	civil_eng	assoc_prof	55 400
Young	male	55	Bonn	german	full_prof	68 000

{ Burnaby, ..., Richmond, Vancouver, Victoria } ⊂ B.C.
{ Hamilton, ..., Toronto } ⊂ Ontario
{ B.C., ..., Ontario } ⊂ Canada
{ Boston, Chicago, ..., Seattle } ⊂ U.S.A
{ Bombay, ..., New Delhi } ⊂ India
{ Beijing, Nanjing, ..., Shanghai } ⊂ China
{ China, India, ..., U.S.A } ⊂ foreign
{ foreign, Canada } ⊂ ANY (Birth-place)
{ secretary, ..., lab_manager } ⊂ staff
{ junior_lecturer, senior_lecturer } ⊂ instructor
{ assist_prof, assoc_prof, full_prof} ⊂ professor
{ staff, instructor, professor } ⊂ ANY (Position)
{ cmpt, electr_eng, ..., civil_eng } ⊂ Applied-Sciences
{ biology, chemistry, ..., physics } ⊂ Science
{ english, german, ..., music } ⊂ Arts
{ Applied-Sciences, Arts, ..., Science } ⊂ ANY (Department)
{ 21 – 30 } ⊂ young
{ 31 – 50 } ⊂ mid-age
{ 51 – 70 } ⊂ old
{ ..., young, mid-age, old } ⊂ ANY (Age)
{ 20000 – 30000 } ⊂ low
{ 30001 – 50000 } ⊂ medium
{ 50001 – 100000 } ⊂ high
{ ..., low, medium, high } ⊂ ANY (Salary)
{ male, female } ⊂ ANY (Sex)

Fig. 2. A concept hierarchy table relevant to the learning task.

TABLE III
THE GENERALIZED RELATION AFTER THE REMOVAL OF THE ATTRIBUTE NAME

Sex	Age	Birth_place	Salary
male	45	Vancouver	63 000
	...	...	...
male	38	Seattle	55 400

Strategy 1: (Generalization on the smallest decomposable components) Generalization should be performed on the smallest decomposable components (or attributes) of a data relation.

Rationale: Generalization is a process of learning from positive examples. Generalization on the smallest decomposable components instead of on composite attributes ensures that the smallest possible chance is considered in the generalization, which enforces the *least commitment principle* (commitment to minimally generalized concepts) and avoids over-generalization. □

The generalization is first performed on each attribute in the initial data relation. We examine the task relevant attributes in sequence. First, there is no higher level concept specified on the first attribute *Name*. Obviously, the attribute should be removed in generalization, which implies that general properties of a *professor* cannot be characterized by the attribute *Name*. A portion of the result relation is shown in Table III. This is based on Strategy 2.

Strategy 2: (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 in the generalization process.

Rationale: This strategy corresponds to the generalization rule, *dropping conditions*, in *learning-from-examples* [13]. Since an attribute–value pair represents a conjunct in the logical form of a tuple, removal of a conjunct eliminates a constraint and thus generalizes the rule. If there is a large set of distinct values in an attribute but there is no higher level concept provided for it, it cannot be generalized using higher level concepts and thus the attribute must be removed. □

In Table III,values in each of the three attributes *Age*, *Birth-place*, and *Salary* can be generalized by substituting the lower level concepts by their corresponding higher level concepts. For example, *Vancouver* can be generalized to *B.C.* and then to *Canada*. Such a substitution is performed attribute by attribute, which is based on Strategy 3.

Strategy 3: (Concept tree ascension) If there exists a higher level concept in the concept tree for an attribute value of a tuple, the substitution of the value by its higher level concept generalizes the tuple. Minimal generalization should be enforced by ascending the tree one level at a time.

Rationale: This strategy corresponds to the generalization rule, *climbing generalization trees*, in *learning-from-examples* [13]. The substitution of an attribute value by its higher level concept makes the tuple cover more cases than the original value and thus generalizes the tuple. Ascending the concept tree one level at a time ensures that the generalization shall follow the least commitment principle and thus reduces chances of over-generalization. □

As a result of concept tree ascension, different tuples may be generalized to the same tuple. A tuple is *redundant* if it is identical to some tuple(s) of the same class in a generalization relation. The removal of redundant tuples reduces the size of the generalized relation. To judge whether an attribute needs to be further generalized, we have Strategy 4.

Strategy 4: (Threshold control on each attribute) If the number of distinct values of an attribute in the target class is larger than the specified generalization threshold value, further generalization on this attribute should be performed.

Rationale: The generalization threshold represents the maximum number of tuples of the target class in the final generalized relation. If one attribute contains more distinct

values than the threshold, the number of distinct tuples in the generalized relation must be greater than the threshold value. Thus the values in the attribute should be further generalized.□

After attribute-oriented ascension of concept trees and the removal of redundant tuples, the total number of tuples in a generalized relation may still be greater than the specified threshold. In this case, further generalization is still required. A strategy has been devised for this further generalization.

Strategy 5: (Threshold control on generalized relations) If the number of tuples of a generalized relation in the target class is larger than the specified generalization threshold value, further generalization on the relation should be performed.

Rationale: Based on the definition of the generalization threshold, further generalization should be performed if the number of tuples in a generalized relation is larger than the threshold value. By further generalization on selected attribute(s) and the elimination of redundant tuples, the size of the generalized relation will be reduced. Generalization should continue until the number of remaining tuples is no longer greater than the threshold value. □

At this stage, there are usually alternative choices at selecting a candidate attribute for further generalization. Criteria, such as the preference of a larger reduction ratio on the number of tuples or on the number of distinct attribute values, etc. can be used for selection. Actually, interesting rules can often be discovered by following different generalization paths to generate several generalized relations for examination, comparison, and selection. This type of discovery corresponds to the fact that different people may learn differently from the same set of examples. Then generalized relations can be examined by users or experts to filter out some trivial rules and preserve interesting ones.

The **final generalized relation** consists of only a small number of tuples, which can be transformed to a simple logical formula. Based on the principles of logic and databases [8], [21], we have evolved Strategy 6.

Strategy 6: (Rule transformation) A tuple in a final generalized relation is transformed to conjunctive normal form, and multiple tuples are transformed to disjunctive normal form.

To incorporate quantitative information in the learning process, a special attribute, *vote*, can be attached to each generalized relation. The **vote** of a tuple t represents the number of tuples in the initial data relation which are generalized to the tuple t in the current generalized relation. The vote of each tuple in the initial data relation is assumed to be one. Notice that two tuples in a generalized relation are **identical** if they have the same corresponding attribute values without considering the special attribute *vote*. Strategy 7 is then obvious.

Strategy 7: (Vote propagation) The value of the vote of a tuple should be carried to its generalized tuple and the votes should be accumulated in the preserved tuple when other identical (and thus redundant) tuples are removed in generalization.

Rationale: Based on the definition of *vote*, the vote of each generalized tuple must register the number of the tuples in the initial data relation generalized to the current one. Therefore, to keep the correct number of votes registered, the vote of each tuple should be carried in generalization, and the vote of a redundant tuple should be added to that of the preserved identical tuple before it is removed from the generalized relation. □

TABLE IV
THE FINAL GENERALIZED RELATION

Sex	Age	Birth_place	Salary	Vote
male	old	Canada	high	20
male	mid-age	Canada	medium	50
female	mid-age	Canada	medium	8
male	mid-age	foreign	medium	21
female	mid-age	foreign	medium	1

Suppose the threshold is set to 5 and the final generalized relation of Example 1 is Table IV. The table corresponds to a rule in the disjunctive normal form. It is derived by *preprocessing*, *attribute removal*, *concept tree ascension*, *vote propagation*, and *threshold control*.

To measure the typicality of each tuple in the characteristic rule, we define *t-weight* in the following.

Definition: Let q_a be a generalized tuple. The ***t*-weight** for q_a is the percentage of the original tuples covered by q_a in the target class. Formally, we have

$$t\_\text{weight} = \text{Votes}(q_a) \Big/ \sum_{i=1}^{N} \text{Votes}(q_i)$$

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

Obviously, the range for t-weight is [0–1]. For example, the t-weight of the first tuple in the Table IV is $20/(20 + 50 + 8 + 21 + 1) = 20\%$. Similarly, the t-weight for the second to the fifth ones are 50, 8, 21, and 1%, respectively. The first tuple implies that *among the professors in Applied Sciences, 20% are male, old, born in Canada and earning high salaries.* The other four tuples can be described similarly. The rule can be represented either: i) in the relational table form by changing *vote* and the corresponding vote values in Table IV to t-weight and the corresponding t-weight values, or ii) in the logical form by associating a corresponding t- weight value with each disjunct.

In general, a quantitative characteristic rule provides the necessary condition of the target class since the condition is derived based on all the facts in the target class, that is, the tuples in the target class must satisfy this condition. However, the rule may not be a sufficient condition of the target class since a tuple satisfying the same condition could belong to another class. Therefore, the rule should be in the form of

$$\forall(x)\text{target\_class}(x) \rightarrow \text{condition}_1(x)[t\colon w_1] \vee \cdots \vee \text{condition}_n(x)[t\colon w_n].$$

The rule indicates that if x is the target_class, there is a possibility of w_i that x satisfies condition$_i$ where i is in $\{1, \cdots, n\}$.

Hence, the final generalized relation of Table IV can be transformed into the logical rule form as 2), where only the first disjunct is shown explicitly, and the others are similar and hence denoted by dots.

2) $\forall x$ [professor(x) $\rightarrow$ ((Sex(x) = male) $\wedge$ (*Age*(x) $\in$ old) $\wedge$ (*Birth_place*(x) $\in$ Canada) $\wedge$ (*Salary*(x) $\in$ high)[t: 20%]) $\vee \cdots$.

The previous discussion can be summarized in the following algorithm

Algorithm 1. *LQCHR—Learning quantitative characteristic rules from relational databases.*

Input:

i) relational database,

ii) a concept hierarchy table,

iii) the learning task specification,

iv) the threshold value (T).

Output: A quantitative characteristic rule of the learning task.

Method:

Step 1: *Collect the task-relevant data by a relational query.*

Step 2: *Call procedure Attribute-Oriented Induction.*

Procedure: *Attribute-Oriented Induction;*

{Suppose that the task relevant relation P consists of a set of attributes, A_i, where $1 \leq i \leq n$. N represents the number of tuples, and d_i represents the number of distinct values of attribute A_i in the target class.}

```
BEGIN
 FOR EACH attribute A_i DO
   WHILE d_i > T DO
   BEGIN
    IF there is no higher level concept
     provided for A_i
    THEN Remove attribute A_i
    ELSE Substitute the values by its
     corresponding minimally generalized
     concept;
    Eliminate redundant tuples (with the
     votes accumulated)
   END
 {Now the number of distinct values of
  each remaining attribute is less than T.}
 WHILE N > T DO
 BEGIN
  Selectively generalize an attribute;
  Eliminate redundant tuples (with the
   votes accumulated)
 END
END. {Attribute-Oriented Induction}
```

Step 3: *Transform the final generalized relation into a logical formula.* □

Notice that the statement, "*selectively generalize an attribute*," in the algorithm indicates that further generalization is based on certain attribute selection criteria implemented in the system, such as user/implementor preference, or better tuple reduction ratio, etc.

Theorem 1: Algorithm LQCHR correctly derives a quantitative characteristic rule for the learning task from a relational database.

Proof: As discussed in Section II, Step I collects the data in the database relevant to the learning task. Based on the discussion in this section, Step 2 generalizes the concept in each attribute by *attribute removal* (Strategy 2) or *concept tree ascension* (Strategy 3), which simulates the generalization process of *learning-from-examples*. Moreover, the specified threshold value ensures that the concept tree ascension process terminates when it reaches the threshold-controlled number of disjuncts (Strategies 4 and 5), and *vote propagation* guarantees association of the correct quantitative information (Strategy 7). Step 3 performs transformation based on the principles of logic and database (Strategy 6). Thus the obtained rule should be the desired result which characterizes the properties of the class. □

TABLE V
THE QUALITATIVE CHARACTERISTIC RULE FOR *Professors* IN *Applied Sciences*

Sex	Age	Birth_place	Salary
male	old	Canada	high
ANY	mid-age	ANY	medium

Learning a qualitative rule can be treated as a special case of learning its quantitative counterpart. A qualitative rule does not associate quantitative information. Thus a qualitative characteristic rule can be obtained by following the same process of learning its quantitative counterpart without the association of the special (quantitative) attribute *vote* in the generalized relations. It can also be derived directly from the final generalized relation by dropping the attribute vote (or t-weight).

It is often possible and desirable to perform further simplification on the final generalized relation. For example, if two tuples are almost identical except for the values of the one attribute, the two tuples can be merged into one by grouping the two values of that attribute using set notation. For example, the second and the third tuples of Table IV, (*male*, *mid-age*, *Canada*, *medium*) and (*female*, *mid-age*, *Canada*, *medium*), share the values in all other attributes except for the first one, *Sex*. The two tuples can be merged into one tuple with a set value {*male*, *female*}. Moreover, since the {*male*, *female*} covers all the possible values of the attribute *Sex*, it can be automatically generalized to ANY and then removed from the conjunction. Similar merging can be performed on the fourth and the fifth tuples. Furthermore, the two merged tuples (*ANY*, *mid-age*, *Canada*, *medium*) and (*ANY*, *mid-age*, *foreign*, *medium*) can be further merged into (*ANY*, *mid-age*, *ANY*, *medium*) because *Canada* and *foreign* cover the whole set of the concept of the attribute "*Birth_Place*." Therefore, the qualitative characteristic rule for *professors* in *Applied Sciences* becomes Table V. If such simplication is performed in the learning of a quantitative rule, corresponding votes should be merged (accumulated) in the merged tuple as in the removal of redundant tuples.

The qualitative characteristic rule can be represented in the logical form as 3). Similar to the corresponding quantitative rule, it represents a necessary condition, which may or may not be a sufficient condition, for x to be a *professor*.

3) $\forall x$(professor(x) $\rightarrow$

((*Sex*(x) = male) $\wedge$ (*Age*(x) $\in$ old) $\wedge$ (*Birth_place*(x) $\in$ Canada) $\wedge$ (*Salary*(x) $\in$ high)) $\vee$ ((*Age*(x) $\in$ mid-age) $\wedge$ (*Salary*(x) $\in$ medium))).

B. Learning Classification Rules

A classification rule discriminates the concepts of the target class from that of the contrasting class(es). In order to extract a classification rule, it is necessary to test, in the generalization process, whether a generalized concept in the target class overlaps with or is covered by one or a set of generalized concepts in the contrasting class(es). Notice that primitive data in both kinds of classes usually reside at the same level (other cases will be discussed in Section IV), and an overlapping test is usually more efficient than a coverage test. Therefore, the concept tree ascension in all of the participating classes should be synchronized in the learning process. We examine such a learning process in detail.

Example 2: Suppose we have the same database as in Example 1, and the task is to *learn a quantitative classification rule for the professors versus the instructors in Applied Sciences relevant to attributes Name, Sex, Age, Birth-place, and Salary.*

The seven strategies studied in the last subsection are generally applicable to the learning of classification rules. However, in order to discriminate the generalized concepts in the target class *professor* from those in the constrasting class *instructor*, we should partition the task relevant data into two portions, one for *professor* and the other for *instructor*, and perform concept tree ascension synchronously on the two partitions. The generalization threshold is the maximum number of tuples in the generalized portion of the *target class only*.

To distinguish tuples shared by both the target class and the contrasting class(es), overlapping tuples should be handled carefully. An *overlapping tuple* is a tuple in the (data or generalized) relation shared between the target class and the contrasting class(es). In general, Strategy 8 handles these cases.

Strategy 8: (Handling overlapping tuples) Overlapping tuples should be marked and such marks should be propagated in the generalization process in the learning of classification rules.

Rationale: Overlapping tuples are the tuples shared by the target class and the contrasting class(es). They represent the features in the target class that are nondistinguishable from the contrasting class(es), and therefore, should be marked for proper handling in the final generalized relation. If two tuples are overlapping, their synchronous concept tree ascension will result in the overlapping of their generalized tuples. Thus the overlapping mark should be propagated to their generalized tuples. □

Suppose the threshold is set to 5 in our example. We drop the attribute *Name* (Strategy 2) and perform generalization on the portion of the target class and that of the contrasting class(es) simultaneously, which is done on the attributes *Age*, *Birth-place*, and *Salary* by iteratively ascending the concept trees (Strategy 3), removing redundant tuples, propagating the votes (Strategy 7), and marking the overlapping tuples (Strategy 8) until the number of unmarked tuples in the target class of the final generalized relation is within the specified threshold (Strategy 5). Suppose the final generalized relation is as depicted in Table VI.

TABLE VI
THE FINAL GENERALIZED RELATION

(Learning Concept)	Sex	Age	Birth_place	Salary	Mark	Vote
professor	male	old	Canada	high		20
	male	mid-age	Canada	medium	*	50
	female	mid-age	Canada	medium		8
	male	mid-age	foreign	medium	*	21
	female	mid-age	foreign	medium		1
instructor	male	young	Canada	low		30
	male	mid-age	Canada	medium	*	15
	female	young	Canada	low		4
	male	mid-age	foreign	medium	*	1

The generalized process can be described in an algorithm, LQCLR (*Learning Quantitative CLassification Rules*) in the following, which is similar to Algorithm LQCHR.

Algorithm 2. *LQCLR—Learning quantitative classification rules from relational databases.*

Input:
i) a relational database,
ii) a concept hierarchy table,
iii) the learning task,
iv) the threshold value (T).

Output: The classification rule of the learning task.

Method:

Step 1: *Collect the task relevant data and partition them according to the learning task specification into two classes: the target class and the contrasting class.*

Step 2: *Invoke procedure attribute-oriented induction.*

Procedure: *Attribute-oriented induction;*

{Suppose the relation relevant to the learning task, P, consists of a set of attributes, A_i, where $1 \leq i \leq n$. Let T be the threshold value, N be the number of unmarked tuples in the target class, and d_i be the number of distinct values of attribute A_i in the unmarked tuples of the target class.}

```
BEGIN
 FOR EACH attribute A_i DO
 BEGIN
  Perform intersection between the target
   and contrasting classes and mark the
   overlapping tuples;
  WHILE d_i > T DO
   IF there is no higher level concept
    of A_i
   THEN remove attribute A_i;
  ELSE BEGIN
     Substitute the values by its corre-
      sponding minimally generalized con-
      cept;
     Mark the newly generalized overlap-
      ping tuples (i.e., those overlapped
      with the tupes in contrasting clas-
      ses); and
     Eliminate identical tuples within
      each class (with the votes accumu-
```

```
      lated);
  END
 END
 {The number of distinct values of each
  remaining attribute in the target class
  is less than T.}
 WHILE N > T DO
 BEGIN
  Generalize the attributes containing
   more distinct values or those with a
   better reduction ratio;
  Mark the newly generalized overlapping
   tuples; and
  Eliminate identical tuples within each
   class (with the votes accumulated);
 END
End. {Attribute-oriented induction}
```

Step 3: *Simplify the generalized relation and transform the final generalized relation into a logical formula.* □

Theorem 2: Algorithm LQCLR correclty learns classification rules from relational databases.

Proof Sketch: Step 1 collects the task relevant data from the database and partitions them into the target class and the contrasting classes based on the discussion in Section II. Step 2 generalizes the concept in each attribute either by "*concept tree ascension*" (Strategy 3) or by "*attribute removal*" (Strategy 1), which simulates the generalization process of *learning-from-examples*. Moreover, the specified threshold value ensures that the process of concept tree ascension terminates when it reaches the threshold controlled number of disjuncts (Strategy 5), and "*handling overlapping tuples*" marks the properties shared by other classes (Strategy 8). Step 3 performs simplification and transformation based on the principles of logic and databases (Strategy 7). Thus the rule so obtained should be the desired result which characterizes the discriminating property of the target class. □

To measure the discriminating behavior of the learned classification rule, we introduce another statistical value, *d-weight*.

Definition: Let q_a be a generalized concept (tuple) and C_j be the target class. The **d-weight** for q_a (referring to the target class) is the ratio of the number of original tuples in the target class covered by q_a to the total number of tuples in both the target class and the contrasting classes covered by q_a. Formally, the d-weight of the concept q_a in class C_j is defined as

$$d\_\text{weight} = \text{Votes}(q_a \in C_j) \Big/ \sum_{i=1}^{K} \text{Votes}(q_a \in C_i)$$

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

The range for d-weight is [0–1]. A high d-weight indicates that the concept is primarily derived from the target class C_j, and a low d-weight implies that the concept is primarily derived from the contrasting class(es).

TABLE VII
THE QUALITATIVE CLASSIFICATION RULE FOR *Professors* IN *Applied Sciences*

Sex	Age	Birth_place	Salary
male	old	Canada	high
female	mid-age	ANY	medium

The d-weight for the first tuple in the target class is $20/(20+0) = 100\%$. Notice that the d-weight for any unmarked tuple is 100%. Accordingly, the d-weights for the second to the fifth tuples are 77, 100, 95, and 100%, respectively. We conclude that *among the professors and instructors in Applied Sciences, if he was born in Canada, is old and earns high salary, he is a professor with a probability of 100%*. Similarly, we can derive the other disjuncts of the quantitative rule.

By associating d-weights, a classification rule provides quantitative criteria to determine the class membership of the data. The quantitative classification rule is represented by the tuples in the target class. It can be represented: i) in the relational table form by changing the *vote* and its associated values in Table VI to d-weight and the corresponding d-weight values, or ii) in the logical form with a d-weight associated with each disjunct.

In general, a quantitative classification rule provides a sufficient condition of the target class since it presents a quantitative measurement of the properties which occur in the target class versus that occurring in the contrasting classes. Therefore, the learned rule should be in the form of

$$\forall(x)\, \text{target\_class}(x) \leftarrow \text{condition}_1(x)[d\colon w_1] \vee \cdots \vee \text{condition}_n(x)[d\colon w_n].$$

The rule indicates that if x satisfies condition_i, there is a possibility of w_i that x is in the target_class, where i is in $\{1, \cdots, n\}$.

Therefore, the final generalized relation of Table VI can be transformed into a logical rule as 4), where only the first disjunct is shown explicitly, and the others are similar, and hence, denoted by dots.

4) $\forall x$[professor(x)$\vee$

(($Sex(x)$ = male) $\wedge$ ($Age(x) \in$ old) $\wedge$ ($Birth\_place(x) \in$ Canada) $\wedge$ ($Salary(x) \in$ high)[d: 100%]) $\vee \cdots$.

Similarly, a qualitative classification rule can be treated as a special case of its quantitative counterpart. To represent a qualitative classification rule, the *vote* information is dropped from the generalized relation(s) and marked tuples in the target class are removed from the rule. The rule can also be obtained directly from the learned quantitative rule. In this case, a tuple with a d-weight not equal to 100% should not be presented in the qualitative rule since it represents the property being overlapped with the contrasting classes. In our example, the qualitative classification rule for *professors* in *Applied Sciences* should be Table VII (with {*Canada, foreign*} merged to *ANY*).

The final generalized relation implies that *among the professors and instructors of Applied Sciences, if he was born in Canada, is old, and earns a high salary*, he is a professor; *if she is mid-aged and earns a medium salary, she is a professor*. It can also be represented in the following logical form.

5) $\forall x$(professor(x) $\leftarrow$
((*Sex*(x) = male) $\wedge$ (*Age*(x) $\in$ old) $\wedge$ (*Birth_place*(x) $\in$ Canada) $\wedge$ (*Salary*(x) $\in$ high)) $\vee$
((*Sex*(x) = female) $\wedge$ (*Age*(x) $\in$ mid-age) $\wedge$ (*Salary*(x) $\in$ medium))).

In general, a qualitative classification rule represents the sufficient condition of the target class since it excludes the properties occurring in the contrasting classes. However, the rule may not be the necessary condition of the target class because it may not cover *all* of the tuples in the target class. Therefore, the learned rule, as presented in rule 5), should be in the form of

$$\forall x \text{ target\_class}(x) \leftarrow \text{condition}(x).$$

When there are no overlapping data discovered (i.e., marked) in the learning process, the learned tuple represents both necessary and sufficient conditions because it covers all of the tuples in the target class but none of the tuples in the contrasting class(es). In this case, the rule should be of the form

$$\forall x \text{ target\_class}(x) \leftrightarrow \text{condition}(x).$$

When both d-weights and t-weights are associated with the same set of tuples, the quantitative classification and characteristic rules can be represented in the same logical rule with the two weights associated with each disjunct. In this case, the bi-directional arrow can be used in the rule representation. That is, it should be in the form of

$$\forall(x)\text{target\_class}(x) \leftrightarrow \text{condition}_1(x)[t: w_1, d: w_1'] \vee \cdots \vee \text{condition}_n(x)[t: w_n, d: w_n'].$$

This form indicates that for i from 1 to n, if x is the target_class, there is a possibility of w_i that x satisfies condition$_i$; and if x satisfies condition$_i$, there is a possibility of w_i' that x is in the target_class.

For example, the quantitative classification rule and characteristic rule in our two examples can be merged into one as shown in 6) where the bi-directional arrow is used and the two weights are associated with each disjunct.

6) $\forall x$ (professor(x) $\leftrightarrow$
((*Sex*(x) = male) $\wedge$ (*Age*(x) $\in$ old) $\wedge$ (*Birth_place*(x) $\in$ Canada) $\wedge$ (*Salary*(x) $\in$ high)[t: 20%, d: 100%]) $\vee$
((*Sex*(x) = male) $\wedge$ (*Age*(x) $\in$ mid-age) $\wedge$ (*Birth_place*(x) $\in$ Canada) $\wedge$ (*Salary*(x) $\in$ medium)[t: 50%, d: 77%]) $\vee$
((*Sex*(x) = female) $\wedge$ (*Age*(x) $\in$ mid-age) $\wedge$ (*Birth_place*(x) $\in$ Canada) $\wedge$ (*Salary*(x) $\in$ medium)[t: 8%, d: 100%]) $\vee$
((*Sex*(x) = male) $\wedge$ (*Age*(x) $\in$ mid-age) $\wedge$ (*Birth_place*(x) $\in$ Foreign) $\wedge$ (*Salary*(x) $\in$ medium)[t: 21%, d: 95%]) $\vee$
((*Sex*(x) = female) $\wedge$ (*Age*(x) $\in$ mid-age) $\wedge$ (*Birth_place*(x) $\in$ Foreign) $\wedge$ (*Salary*(x) $\in$ medium)[t: 1%, d: 100%])).

IV. Variations of the Learning Algorithms

A. Handling Noise and Exceptions in Learning

Many data-driven learning algorithms assume that learning is performed in a *noise-free* or *exception-free* environment [12], [14]. Such an assumption may not be realistic in database applications. Usually, a generalized concept may cover a majority of data but cannot cover some special data in the database because of the diverse distribution of data, misclassification or inaccurate measurement of data. Such special kinds of data are called *exceptions* or *noise*.

Many techniques have been developed in machine learning to cope with noise and exceptions [4], [12], [17]. A rule which excludes noise and exceptions is called an **approximate rule.** Since quantitative rules incorporate quantitative information in the learned rules, prime rules can be extracted easily from quantitative rules.

First, we examine the extraction of an *approximate characteristic rule* from a quantitative characteristic rule. The t-weight information carries database statistics and supports the pruning of scattered data. A high t-weight implies that the concept is induced from the majoirty of data, and a low t-weight implies that the concept is derived from some rare, exceptional cases. By pruning low t-weight tuples (disjuncts), the final generalized relation (or rule) characterizes the majority number of facts in the database.

In practice, we may specify a ***t*- threshold** to prune the low t-weighted tuples in the learning process. For example, in the final generalized rule of Table IV, the t-weight of the third tuple (*female*, *mid-age*, *Canada*, *medium*) is 8% and that of the fifth tuple (*female*, *mid-age*, *foreign*, *medium*) is 1%. If the pruning threshold is set to 5%, the fifth tuple should be dropped. If the threshold is increased to 10%, both the third and the fifth tuples should be removed from the prime characteristic rule.

Then we examine the extraction of am *approximate classification rule* from a quantitative classification rule. The d-weight of a nonoverlapping tuple is always 1. A high d-weight (with the value close to 1) indicates that the tuple is primarily generalized from the original tuples in the target class with only some exceptional cases from the contrasting class(es); a low d-weight (with the value close to 0) indicates that it is primarily from the contrasting class(es) with only some exceptional cases from the target class; and a medium d-weight indicates that the generalization is from a relatively even distribution between the target class and the contrasting class(es). Since only tuples with high discriminating behavior are able to discriminate one class from others, only the high d-weighted tuples should count in the classification rule.

In practice, we may specify a ***d*-threshold** to prune the marked tuples with low and medium d-weights in the quantitative classification rule. For example, in Table IV, the d-weight of the second tuple is 77% and that of the fourth tuple is 95%. If the d-threshold is set to 90%, the second tuple should be removed from the approximate classification rule. The fourth tuple (*male*, *mid-age*, *foreign*, *medium*), though marked, is preserved in the classification rule because only about 5% of its original tuples are in the contrasting class.

Both t-threshold and d-threshold are called **pruning thresholds** since they are used to prune exceptional and noisy data in generalization. Pruning thresholds can also be tuned interactively by users/experts in the learning process as the tuning of the generalization threshold discussed in Section III. More sophisticated statistical techniques can be applied to measure the correlations between the generalized tuples and the learning classes [16]; we do not consider them here.

B. Incremental Learning on Database Updates

A flexible database learning technique should allow learning to be performed incrementally on database updates [11]. Incremental learning avoids restarting the costly learning process from the beginning on database updates.

Using the quantitative information learned from a database, incremental learning can be performed efficiently and effectively on database updates. Assume that the database stores the final (quantitative) generalized relation of a learning task. We consider an example.

Example 3: Suppose a new tuple, (*Watt, female, 32, Hamilton, cmpt, assist_prof, 45 500*), is inserted to the data relation of Table II, and the learning task is the same as that of Example 2. Incremental learning should be performed by modifying the learning result of Table VI instead of restarting the entire learning process on the newly updated database. It is performed as follows.

First, the value of the attribute which serves as a classifier of the learning task determines the class to which the newly inserted tuple belongs. For example, "Position = *assist_prof*" indicates that the new tuple is in the class of *professor*. If a newly inserted tuple belongs to neither the target class nor the contrasting class, such as the newly arrived person is neither a professor nor an instructor but a secretary, the previous learning result should not be affected.

Then incremental learning proceeds as follows. The attribute which does not have corresponding attribute entries in the final generalized relation should be removed from the tuple, which corresponds to the strategy *attribute removal*. Other attribute values of the inserted tuple should be generalized to the same level of the concepts as those in the final generalized relation and inserted into the generalized relation. In our example, the tuple is generalized to (*female, mid-age, Canada, medium*) and inserted to the class of *professor*. Since this generalized tuple is the same as the third tuple of Table VI, the effect is just an increment of the vote of the third tuple by 1. The corresponding t-weight and d-weight in the learned quantitative rule should be updated accordingly. □

In general, when tuples are inserted into the database, each attribute of the newly inserted tuples is generalized to the same concept level as those in the original final generalized relation. The newly generalized tuples are merged with the final generalized relation. If they have identical tuples in that relation, the effect is simply incrementing the votes of the identical tuples. Otherwise, they become the new tuples in the final generalized relation. If the size of the newly formed generalized relation is larger than the threshold value, further generalization should be performed on it to derive a new final generalized relation.

Incremental learning can also be performed on database deletions. The deletion of a tuple relevant to the learning result can be implemented by decrementing the vote of the corresponding generalized tuple by one in the final generalized relation. However, the deletion of some sensitive tuples or the deletion of a substantial number of tuples may require restarting the learning process from the beginning because a vote decrement is not equivalent to the reverse of concept tree ascension. If all of the concepts are retained at the levels of concepts in the old generalized relation, this may lead to an overly generalized relation. In such cases, learning should be performed over again on the new database to keep the discovered knowledge up-to-date.

C. Learning with Other Kinds of Concept Hierarchies

Our previous discussion assumes that the concept hierarchy of each attribute, if any, forms a balanced tree. Thus generalization on each attribute can be performed synchronously to generalize the same lower level concepts of an attribute to the same higher level ones.

However, a concept hierarchy could be an unbalanced concept tree, or primitive data may reside at different levels of a concept tree. In such cases, the same level of concepts could be reached at different generalization stages by synchronous concept tree ascension, which may result in incorrect generalization. This problem can be solved by checking whether a generalized concept covers a concept which resides at a level lower than the current one in the concept tree. If it does, the covered concept can be substituted by the covering one, that is, directly ascending that branch of the tree several levels higher to bring concepts at different levels to the same level. Then the same learning algorithms can be performed correctly and efficiently.

V. A Comparison with Other Learning Algorithms

It is interesting to compare our technique with those developed in traditional *machine learning* research [13], [14]. In general, our method adopts the concept tree ascending technique which follows the idea of *the version space method*, a typical method of *learning-from-examples* [5], [14], [15]. However, our method takes advantages of the characteristics of relational database systems and provides many distinct features.

First, most *learning-from-examples* algorithms [5], [13], [14] learn classification rules from both positive and negative examples. They perform generalization using positive instances and specialization using negative instances. Unfortunately, negative examples are not stored in relational databases explicitly. Thus we have to rely on the generalization process, that is, the least commitment generalization and threshold control. In the learning of classification rules, data in the contrasting classes are used to exclude features of the target class which are shared by the contrasting classes. This can be viewed as a specialization process. However, this specialization process is performed by first generalizing tuples in both the target class and the contrasting class(es) and then excluding the overlapping tuples. The rule so learned may not cover all of the positive examples, that is, it is only a sufficient condition but may not be a necessary condition. This specialization process is different from that of *learning-from-examples* which learns both necessary and sufficient conditions of a concept.

Secondly, the algorithm may learn disjunctive rules, which provides additional flexibility over many *learning-from-example* algorithms. Moreover, qualtitative rules can be extracted from the results of learning quantitative rules. By

incorporating quantitative information in the learning process, our method handles noise and exceptions elegantly.

The major benefit of our method in comparison with the version space method is processing efficiency. The version space method adopts tuple-oriented generalization. In contrast, our method adopts the attribute-oriented generalization which treats a concept hierarchy of each attribute as a factored version space and performs generalization on individual attributes. 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 factorized version space should be $p \times k$. However, the size of the unfactorized version space for the same concepts should be p^k [20]. The search space for attribute-oriented generalization is much smaller than the one for tuple-oriented generalization.

Similar arguments hold for the attribute-oriented method in comparison with other tuple-oriented approaches [13], [14]. An attribute-oriented generalization requires the testing of redundant tuples in processing, which is performed after the generalization of all the values on each attribute. In contrast, a tuple-oriented approach requires the testing for concept coverage, which should be performed after each generalization on a single attribute value of a tuple. Since there are a huge number of possible combinations in such testing, inefficient algorithms evolve when operating in large databases [10].

The efficiency of the attribute-oriented generalization can also be demonstrated by analyzing its worst-case time complexity. Suppose there are N tuples in the database which are relevant to the learning task, A attributes for each tuple, and H levels for each concept tree, the time complexity in the worst case is analyzed as follows. For each attribute, the time for substituting the lower level concepts by the higher level concepts is N, and the time for checking redundant tuples is $N \log N$. Since the height of the concept tree is H, the time spent on each attribute is at most $H \times (N + N \log N)$. Obviously, the upper bound of the total time for processing A attributes is $A \times H \times (N + N \log N)$. In general, A and H are much smaller than N in a large database. Therefore, the time complexity of our approach is $O(N \log N)$ in the worst case, which is more efficient than the tuple-oriented generalization [18].

Another obvious advantage of our approach over many others 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 relational systems. While most learning algorithms suffer from inefficiency problems in the large database environment [6], [13], our approach provides an efficient method for learning in such databases.

Attribute-oriented induction was first developed in our previous work [1], [2]. The technique studied here is a further development of our techniques for learning quantitative rules. Quantitative information provides us with informative rules and facilitates learning in the presence of noise and exceptions.

TABLE VIII
THE NEW FINAL GENERALIZED RELATION

Age	Salary	Vote
old	high	20
mid-age	medium	80

Our previous study of learning qualitative rules can be treated as a special case of learning quantitative rules.

VI. APPLICATION OF LEARNING RESULTS

Since the knowledge rules discovered in a database are based on a large set of data, they represent important knowledge about data in the database. The learned rules enrich our understanding of the general properties of data and help discover interesting relationships among data in the database. Therefore, it represents an important technique for knowledge acquisition in databases.

Interestingly, many more rules can be derived from the quantitative rules of a database. First, qualitative rules can be extracted from the learned quantitative rules, by dropping the quantitative measurements t- weight(s) and d-weight(s), as we discussed above. Secondly, approximate rules, which exclude noise and exceptions, can be extracted from the learned quantitative rules by pruning the tuples which are below the specified pruning thresholds. Moreover, rules relevant to a subset of the previously studied set of attributes can often be extracted directly from the previously learned rules. We examine one such example.

Example 4: Suppose the learning task is to characterize the *professors* in *Applied Sciences* relevant to the attributes *Age* and *Salary* only.

The learning task is almost the same as the task posed in Example 1, except that it is relevant to a subset of previously studied set of attributes. Instead of starting the learning process from the beginning, the rule can be extracted directly from the learning result, that is, the final generalized relation, Table IV. The only processing to be performed is to project the irrelevant attributes *Sex* and *Birth-place* and remove redundant tuples, which derives the final generalized relation, Table VIII. That is, *20% professors in Applied Sciences are old with high salaries and 80% of them are mid-aged with medium salaries.*

Another important application of generalized rules and concept hierarchies is to answer queries involving concepts at different levels of abstraction. Although a relational database stores a large amount of data, it cannot answer queries involving concepts at any level higher than the level of the primitive data. For example, a simple query,"*how many professors are in Applied Scineces?*" cannot be answered since the database (Table II) knows neither "professor" (but "assist_prof," "full_prof," etc.) nor "Applied Sciences" (but "electr_eng," "computing science," etc.) A query, "*describe the characteristics of professors in Applied Sciences?* will be more challenging. With the help of generalized rules and concept hierarchies, such queries can be handled naturally. Moreover, queries involving quantitative or statistical information can be answered effectively and efficiently using the extracted quantitative rules. A database user may view data at different

levels of abstraction and inquires concepts at mixed levels, which enhances the usefulness and flexibility of the database.

Furthermore, discovered knowledge benefits semantic query optimization in databases. Previous studies have shown that integrity constraints and deduction rules can be used for semantic query optimization [3]. Actually, induced rules can be used for such optimization as well. For example, a query, "*how many professors in Applied Sciences were born in foreign countries?*" can be answered directly by examining only the final generalized relation. Since generalized rules are extracted from large amounts of data, they represent certain data semantics and characteristics. Therefore, queries involving such semantic information can be processed using induced rules naturally.

VII. Conclusions

We presented an efficient, attribute-oriented induction method for data-driven discovery of quantitative rules in relational databases. Based on the information about data relevance, expected rule forms, and concept hierarchies, the attribute-oriented induction method integrates database operations with the learning process and provides a simple and efficient way for knowledge discovery in large databases.

We studied in detail the method for learning two kinds of quantitative rules, characteristic rules and classification rules. Learning qualitative rules can be treated as a special case of learning quantitative rules. Moreover, by incorporating quantitative information, approximate rules which exclude noise and exceptions can be extracted and incremental learning can be performed efficiently on database updates. Thus learning quantitative rules provides us with a powerful mechanism of learning various kinds of knowledge rules from relational databases.

A preliminary implementation of our approach has resulted in a prototyped database learning systems, DBLEARN. A high level interface has also been constructed for the specification of learning tasks, conceptual hierarchies, and thresholds as well as for communication with users in the learning process. Experiments with DBLEARN on the Natural Sciences and Engineering Research Council of Canada Research Grant Information Database have shown great promise for data-driven discovery of knowledge rules in relational databases. The further development of database learning techniques and the application of the techniques to deductive database systems and rule-based expert systems are interesting topics for the future research.

References

[1] Y. Cai, N. Cercone, and J. Han, "Attribute-oriented induction in relational databases,"in G. Piatetsky-Shapiro and W. J. Frawley, Eds. *Knowledge Discovery in Databases*. Menlo Park, CA: AAAI/MIT, 1990, pp. 213–228.

[2] ——, "An attribute-oriented approach for learning classification rules from relational databases," in *Proc. 6th Int. Conf. Data Engineering*, Los Angeles, CA, Feb. 1990, pp. 281–288.

[3] 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*, San Francisco, CA: Morgan Kaufmann, 1988, pp. 243–274.

[4] K. C. C. Chan and A. K. C. Wong, "A statistical technique for extracting classificatory knowledge from databases," in G. Piatetsky-Shapiro and W. J. Frawley, eds., *Knowledge Discovery in Databases*. Menlo Park, CA: AAAI/MIT, 1991, pp. 107–124.

[5] P. Cohen and E. A. Feigenbaum, *The Handbook of Artificial Intelligence (Vol. III)*. San Francisco, CA: William Kaufmann, 1983.

[6] T. G. Dietterich and R. S. Michalski, "A comparative review of selected methods for learning from examples," in R. S. Michalski *et al.*, Eds., *Machine Learning: An Artificial Intelligence Approach, Vol. 1*. San Francisco, CA: Morgan Kaufmann, 1983, pp. 41–82.

[7] D. Fisher, "Improving inference through conceptual clustering," in *Proc. 1987 AAAI Conf.*, Seattle, WA, July 1987, pp. 461–465.

[8] H. Gallaire, J. Minker, and J. Nicolas, "Logic and databases: A deductive approach," *ACM Comput. Survey*, vol. 16, no. 2, pp. 153–185, 1984.

[9] M. Genesereth and N. Nilsson, *Logical Foundations of Artificial Intelligence*. San Francisco, CA: Morgan Kaufmann, 1987.

[10] D. Haussler, "Quantifying the inductive bias in concept learning," in *Proc. 1986 AAAI Conf.*, Philadelphia, PA, Aug. 1986, pp. 485–489.

[11] D. Kulkarni and H. A. Simon, "The process of scientific discovery: The strategy of experimentation," *Cognitive Sci.*, vol. 12, pp. 139–175, 1988.

[12] M. V. Manago and Y. Kodratoff, "Noise and knowledge acquisition," in *Proc. 10th Int. Joint Conf. Artificial Intelligence*, Milan, Italy, 1987, pp. 348–354.

[13] R. S. Michalski, "A theory and methodology of inductive learning," in R. S. Michalski *et al.*, Eds., *Machine Learning: An Artificial Intelligence Approach, Vol. 1*. San Francisco, CA: Morgan Kaufmann, 1983, pp. 83–134.

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

[15] T. M. Mitchell, "Version spaces: A candidate elimination approach to rule learning," *Proc. 5th Int. Joint Conf. Artificial Intelligence*, Cambridge, MA, 1977, pp. 305–310.

[16] G. Piatetsky-Shapiro, "Discovery of strong rules in databases," in G. Piatetsky-Shapiro and W. J. Frawley, eds., *Knowledge Discovery in Databases*. Menlo Park, CA: AAAI/MIT, 1991, pp. 229–238.

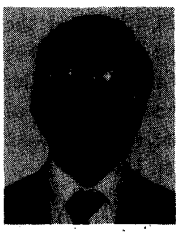
[17] J. R. Quinlan, "The effect of noise on concept learning," in R. S. Michalski *et al*, Eds., *Machine Learning: An Artificial Intelligence Approach, Vol. 2*. San Francisco, CA: Morgan Kaufmann, 1986, pp. 149–166.

[18] S. J. Russell, "Tree-structured bias," in *Proc. 1988 AAAI Conf.*, Minneapolis, MN, Aug. 1988, pp. 641–645.

[19] M. Stonebraker, *Readings in Database Systems*. San Francisco, CA: Morgan Kaufmann, 1988.

[20] D. Subramanian and J. Feigenbaum, "Factorization in experiment generation," in *Proc. 1986 AAAI Conf.*, Philadelphia, PA, Aug. 1986, pp. 518–522.

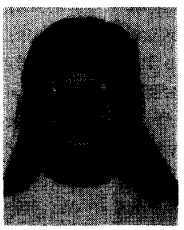
[21] J. D. Ullman, *Principles of Database and Knowledge-Base Systems, Vol. 1*. Palo Alto, CA: Computer Science, 1988.



Jiawei Han received the M.Sc. degree in 1981 and the Ph.D. degree in 1985, both in computer sciences from the University of Wisconsin, Madison.

He was an Assistant Professor with Northwestern University from 1986 to 1987. Presently, he is an Associate Professor with Simon Fraser University, Canada. His current research interests include deductive database systems, knowledge-base systems, logic programming, knowledge discovery in databases, spatial databases, and artificial intelligence.

Dr. Han is a member of the Association for Computing Machinery and the Association for Logic Programming.



Yandong Cai received the B.Sc. degree in 1977 and the M.Sc. degree in 1983 in China. She received the M.Sc. degree in computing science from Simon Fraser University in 1989.

Her current research interests include machine learning, knowledge discovery in databases, and expert database systems.

Nick Cercone (S'72–M'75) for a photograph and biography, please see page 2 of this issue.