# 15 Metaqueries for Data Mining

Wei-Min Shen, USC Information Sciences Institute
KayLiang Ong, MCC
Bharat Mitbander, Tandem Computers
Carlo Zaniolo, UCLA Computer Science Department

**Abstract**

This chapter presents a framework that uses metaqueries to integrate inductive learning methods with deductive database technologies in the context of knowledge discovery from databases. Metaqueries are second-order predicates or templates, and are used for (1) Guiding deductive data collection, (2) Focusing attention for inductive learning, and (3) Assisting human analysts in the discovery loop. We describe in detail a system that uses this idea to unify a Bayesian Data Cluster with the Logical Data Language (*LDL*++), and show the results of three case studies, namely, discovering regularities from a knowledge base, discovering patterns and errors from a large telecommunication database, and discovering patterns and errors from a large chemical database. The patterns discovered using metaqueries are implication rules with probabilities. These rules can link information from many tables in databases, and they can be stored persistently for multiple purposes, including error detection, integrity constraints, or generation of more complex metaqueries.

## 15.1 Introduction

Recent progress in knowledge discovery from databases (Cercone and Tsuchiya 1993; Piatetsky-Shapiro 1993) has shown that inductive hypothesis generation, deductive hypothesis verification, and human intituation, are all crucial components for an effective discovery system. Inductive learning is essential for generating hypotheses from data automatically, deductive database technology is a natural tool for gathering evidence in support of existing hypotheses, while human intuition (which may be inspired by the results of machine discovery) is necessary for generating and selecting the most promising hypotheses in a practical manner. However, the integration of these three components are still poorly supported by today's discovery systems. Most systems simply provide a set of tools (deductive or inductive alike) for humans to choice at arbitration, and once a tool is selected, no attempt is made to exploit the synergy between these three intrinsically related discovery "engines."

In this chapter, we propose an approach to the integration via second-order predicates called *metaqueries*. A metaquery is in essence a second-order template that specifies the type of patterns to be discovered. For example, let $P$, $Q$ and $R$ be variables for predicates, then a metaquery

$$P(X,Y) \wedge Q(Y,Z) \Rightarrow R(X,Z)$$

specifies that the patterns to be discovered are transitivity relations $p(X,Y) \wedge q(Y,Z) \Rightarrow r(X,Z)$, where $p$, $q$, and $r$ are specific predicates. One possible result of this metaquery is the pattern

$$citizen(X,Y) \wedge officialLanguage(Y,Z) \Rightarrow speaks(X,Z)$$
with a probability (say 0.93),

where *citizen, officialLanguage*, and *speaks* are relations that bind to $P$, $Q$, and $R$, respectively, in the current database.

In general, a metaquery can be viewed as a two-part specification: the left-hand side specifies a constraint on how data should be prepared, and the right-hand side specifies an action to be applied on the prepared data. For example, the left-hand of the above example, when $P$ and $Q$ are bound to specific predicates $p$ and $q$ respectively, is a constraint on how to fetch those data pairs $(X,Z)$ that satisfy $p(X,Y) \wedge q(Y,Z)$. The right-hand of that example, when $R$ is bound to a specific predicate $r$, is interpreted as an action that computes the ratio of $(X,Z)$ pairs, among those returned by the left-hand side, that satisfy $r(X,Z)$. (In fact, the precise syntax for the right-hand side of the above metaquery is em RuleStrengthComputer($R(X,Z)$), where em RuleStrengthComputer is an action that performs the described procedure.)

At the present, the framework supports the following four basic metaquery actions:

- *RuleStrengthComputer*($r, [X, Z, ...]$), where $r$ is a predicate and [X, Z, ...] is a tuple of variables for a given set of data. This action returns an expression r(X,Z,...) along with a number (such as the probability 0.93 in the above example) indicating the ratio of the given data tuples that satisfy the predicate r(X,Z,...).
- *Plotter*($[X, Z, ...]$). This action simply plots the given set of data.
- *Classifier*($[X, Z, ...]$). This action uses the CDL incremental learning algorithm (Shen 1994) to return a set of class descriptions that best classify the given data.

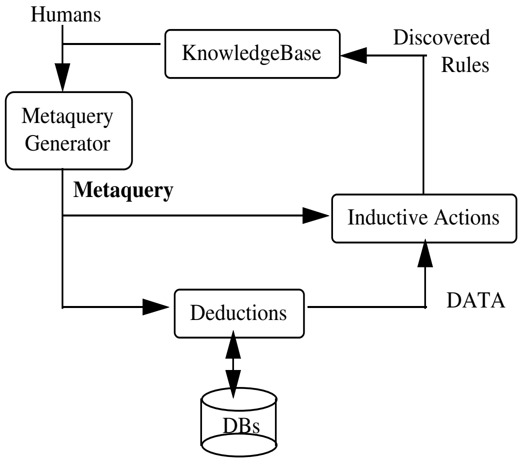


**Figure 15.1**
The Metaquery Framework and The Discovery Loop

- $Cluster([X, Z, ...])$. This action uses a Bayesian-based Cobweb clustering algorithm (Fisher 1987; Shen 1994) to return a set of cluster descriptions that best cluster the given data.

As shown in Figure 15.1, metaqueries serve as the link between the inductive and deductive aspects of knowledge discovery, effectively facilitating a deductive-inductive-human discovery loop. Metaqueries outline the data collecting strategy for the deductive part of the loop; they serve as the basis for the generation of specific queries, which are obtained by instantiating the variables in the metaqueries with values representing tables and columns in the database of interest. These instantiated queries are then run against the database to collect relevant data. Users can either type their metaqueries directly, or have the system generates some initial metaqueries automatically. Furthermore, metaqueries also serve as a generic description of the class of patterns to be discovered and help guide the process of data analysis and pattern generation in the inductive part of the loop. The patterns discovered from the database adhere to the format of the current metaquery. Notice that, unlike many other discovery systems, patterns discovered using metaqueries can link information from many tables in databases. Furthermore, these patterns are all relational, while most machine learning systems can only learn propositional patterns.

Metaqueries can be specified by human experts or alternatively, they can be automatically generated from the database schema. Either way, they serve as a very important interface between human "discoverers" and the discovery system. Using metaqueries, human experts can focus the discovery process onto more profitable areas of the database; the system generated metaqueries provide valuable clues to the human expert regarding good start points for the database searches and also serve as the evolutionary basis for the development of user specified metaqueries more attuned to the discovery goals as envisaged by the human expert.

The above ideas are implemented in a system called the *Knowledge Miner* (KM). Up to date, the induction component of KM is an unsupervised Bayesian Data Cluster (Shen 1994), and the deduction component of KM is a state of the art deductive database technology called *LDL++* (Naqvi and Tsur 1989, Arni, Ong, Tsur, and Zaniolo 1993), which is well suited for both knowledge representation and data querying. Under this implementation, the patterns discovered under metaqueries are actual *LDL++* rules, and their usage is very flexible. Used as queries, they can retrieve the actual data that satisfy or violate a particular rule; Used as integrity constraints, they can alert or filter future "illegal" transactions; Used as logical rules, they can help to generate new metaqueries.

In the rest of this chapter, Section 2 gives an overview of the KM framework and its various components. Sections 3 and 4 describe in detail the *LDL++* language and the unsupervised Bayesian Data Cluster, respectively. Sections 5 to 7 present three case studies in the application of the metaquery approach to knowledge discovery in a knowledge base, a telecommunication database, and a chemical database respectively. Finally, the chapter is concluded with a set of open problems associated with this approach and our plan for future work.

## 15.2 Knowledge Miner: A MetaQuery-Driven Framework

In this section, we give an overview of the metaquery driven framework, and a specific system called "Knowledge Miner" that implements the ideas described above.

Figure 15.2 illustrates the configuration of Knowledge Miner. Given a database and its schema, the metaquery generator (to be described later) suggests a series of valid metaqueries. Users can either pick some of the suggested metaqueries or they can specify their own metaqueries to be run.

On being provided with a metaquery as input, the instantiator generates a set of corresponding *LDL++* queries by instantiating the predicate variables in the metaquery with relevant table names and column names as appropriate. These *LDL++* queries are then run against the database to gather the data in the desired fashion. The resulting data is fed to the Bayesian Data Cluster, which classifies the data into classes and then passes the classification information to the Probabilistic Rule Constructor. The

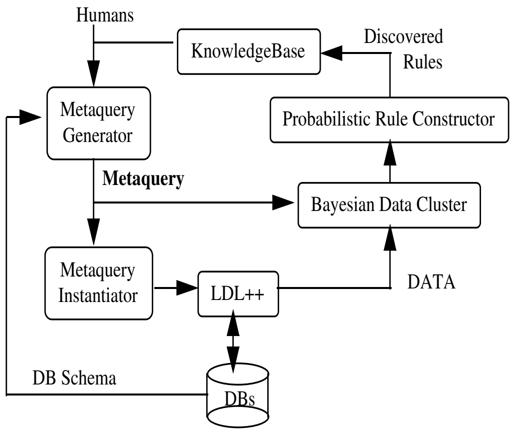


**Figure 15.2**
Knowledge Miner: A MetaQuery-Driven Framework

constructor then builds relations and patterns in the format specified in the current metaquery. The classification information produced by the Bayesian Data Cluster can also be stored persistently in a database.

Consider, for example, a database that has 4 tables, each with an arity of 2. Suppose that table1 and table2 specify ingredients of chemical compounds, and table3 and table4 specify two properties of compounds, $P_1$ and $P_2$, respectively. (These information can be presented to KM by experts or be extracted automatically from the schema by the metaquery generator.) If we are interested in finding the relation between ingredients and properties, then a proper metaquery is as follows:

$$Ingredients(X, c_1, c_2) \wedge Property(X, Y) \Rightarrow Cluster(Y)$$

where $X$ and $Y$ are variables, $c_1$ and $c_2$ are constants, and *Cluster* is the inductive action to be performed on $Y$. Given this metaquery, the instantiator instantiates it into a series of *LDL*++ queries with variables bound to appropriate table and column names. In the current example, such *LDL*++ queries are:

$$Q1(Y) \leftarrow table1(X, c1),\ table2(X, c2),\ table3(X, Y)$$
$$Q2(Y) \leftarrow table1(X, c1),\ table2(X, c2),\ table4(X, Y)$$

Each rule is run against the database, and a set of values that satisfy the constraints on the right-hand side of the rule is fed into the Bayesian Data Cluster. The cluster classifies these values into a set of classes, each with a mean value, a variance, and a likelihood. For example, when $c_1$='BX89' and $c_2$='GF102', the $Y$ values that satisfy the first rule may be classified into two classes: $(m_1, v_1, l_1)$ and $(m_2, v_2, l_2)$, where $m_i$ are mean values, $v_i$ are variances, and $l_i$ are the likelilhood of the class.

These clustering results are then fed into the Rule Constructor, which uses the current metaquery as a template to generate a final rule. For example, suppose the two classes above are: ($m_1$=2.4, $v_1$=3.5, $l_1$=0.97) and ($m_2$=202.0, $v_2$=0.5, $l_2$=0.03), then the final rule constructed using the metaquery may look like this:

> ingredients='BX89'&'GF102' $\wedge$ property=$P_1$ $\Rightarrow$ Clusters($P_1$)={(2.4,3.5, 0.97), (202.3, 0.5, 0.03)}

This rule says for all compounds that contains ingredients 'BX89' and 'GF102', the majority values of their poperty $P_1$ are near 2.4, but there are few values scattered around 202.3.

We now discuss each of the components of KM in detail. In particular, the following two sections cover the *LDL*++ language and the Bayesian Data Cluster. The Metaquery Generator will be discussed after the presentation of experimental results obtained using KM.

## 15.3 The *LDL*++ Deductive Database System

The *LDL*++ system is a deductive database system based on the integration of a logic programming system with relational database technology. It provides a logic-based language that is suitable for both database queries and knowledge representation. More details on the *LDL*++ system and language can be found in (Naqvi and Tsur 1989; Arni, Ong, Tsur, and Zaniolo 1993; Zaniolo 1994). In this section, we will briefly describe some of the salient aspects of the *LDL*++ system as relevant to KM and also highlight the benefits of using *LDL*++ in the Knowledge Miner framework.

The *LDL*++ query language is based on Horn clause logic (Lloyd 1984; Apt 1990) and a *LDL*++ program is essentially a set of declarative rules. For example, the following rules

$$\begin{aligned} ancestor(X,Y) &\leftarrow parent(X,Y). \\ ancestor(X,Y) &\leftarrow ancestor(X,Z), parent(Z,Y). \end{aligned}$$

specify that a new relation `ancestor/2` can be defined based on the relation `parent/2`. `X`, `Y` and `Z` are variables and `ancestor/2` and `parent/2` are predicate symbols, both with arity of 2. By declarativeness, we mean that the ordering between the rules is unimportant and will not affect the results returned. Deduction of all values of the ancestor relation is achieved through an iterative bottom-up execution model.

The language supports a rich set of complex data types such as complex objects, lists and sets in addition to other basic types such as integer, real and string. Examples of each of these complex types are `rectangle(1,2)`, `[1,2]` and $\{1,2\}$ respectively.

With such a rich set of data types, the *LDL++* language is well suited for representing knowledge discovered from databases. Furthermore, complex constraints in the metaqueries can be easily specified in the language. Through the metalevel query facility in *LDL++*, constraints represented in the form of data can be used to generate rules at run-time.

From the database query perspective, in comparison to SQL, the *LDL++* language extends the ability of KM to impose more complex constraints on the data it retrieves beyond what is possible in SQL. This is illustrated by the recursive rules that define `ancestor/2` - such constraints cannot be specified in SQL directly. Moreover, cascading or nested queries can be easily represented in LDL++ in the form of rules.

Systemwise, the *LDL++* system was designed with an open architecture. Even though it was designed independently of KM needs, it meets many of the KM demands. It is "opened" to procedural languages such as C/C++ in two ways: It provides an Application Programming Interface (API) that allows applications to drive the system and, an External Function Interface (EFI) that allows C/C++ routines to be imported into the inference engine. It is also "opened" to external databases such as Sybase, Oracle, Ingres and DB2 through its External Database Interface (EDI).

Referring to Figure 15.2, we can observe that the *LDL++* system has to interface with the Instantiator, the Database and the Bayesian Data Cluster. It interfaces with the Instantiator throught its API, the Bayesian Data Cluster with its EFI and the Database with its EDI.

Both, tables in the external databases and C/C++ interface routines, are modeled as predicates through its EDI and EFI respectively. As a result, these external resources are transparent to the inference engine and the KM can plug in a different database or another learning algorithm with ease, without making any changes to the overall implementation. The EDI facilitates the retrieval of data as well as the archiving of statistical metadata into persistant storage.

The EDI are designed to provide access data from multiple, heterogeneous databases simultaneously in one single query. Furthermore, it also provides the capability to fuse the data from the various and combine into one single view for knowledge discovery. This empowers KM to discover relationships between data from different sources. Again, the data access is transparent to KM and no changes are required to take advantage of this capability.

The following example shows the *LDL++* schema declaration to access an external Sybase relation `employee` in the database `payroll` running on the server `server1`.

**Example 1** Schema Declaration to external Sybase server.

```
database({
        sybase::employee(NAME:char(30),SALARY:int, MANAGER:char(30))
                from server1
                use payroll
                user_name 'john'
                application_name 'downsizing'
                interface_filename '/tmp/ldl++/demo/interfaces'
                password nhoj
          } ).
```

Predicates on external relations in rules are translated to standard SQL queries that are applicable to all relational databases with an SQL interface. The *LDL++* system generates SQL queries that off-load join, select, and project operations (and negation and aggregate operations as well) to the external database server. This is done using the rewriting optimizations that collapse one or more rules using external relations and comparison predicates into one single SQL statement. To illustrate, consider the following example

**Example 2** SQL Generation

```
   expensive_employee(Name) <-
                employee(Name, Salary1, Manager),
                Salary1 > 75000,
                employee(Manager, Salary2, _),
                Salary1 > Salary2.
```

The rule in the example defines expensive employees as those who make over 75,000 and more than their managers. The *LDL++* compiler will collapse the body literals and transform them into one SQL node as shown below:

```
        expensive_employee(Name) <- sql_node(Name).
```

Where the SQL query corresponding to `sql_node`, that is sent to the server is:

```
SELECT  employee_0.NAME
FROM    employee employee_0, employee employee_1
WHERE   employee_0.SALARY > 75000 AND
      employee_1.NAME = employee_0.MANAGER AND
      employee_0.SALARY > employee_1.SALARY
```

Consequently, access to the external database via *LDL++* is as efficient as any queries written directly in SQL. Rules with negated goals are also supported and implemented via the `NOT EXIST` construct of SQL.

Last but not least, once the statistical data is derived and high-probability rules are created, they are represented directly in *LDL++* for enforcing the integrity of the database.

## 15.4 Bayesian Data Clustering

Since most our applications involve unsupervised data classification, we have chosen to integrate a cluster as our first inductive action. To classify the retrieved data, we use a *conceptual clustering* approach based on the Cobweb algorithm (Fiher 1987) but adapted to use discriminant functions derived from Bayesian probability theory (Jaynes 1993; Shen 1994). We feel that the latter lends a stronger mathematical basis to the evaluation process.

The Cobweb algorithm facilitates incremental, unsupervised learning of concept hierarchies from attribute-based instances. The concepts formed represent nodes in a concept tree, with each non-leaf node further partitioned into its more specific children nodes. Thus, more general concepts are located near the root of the tree and more specific concepts are located towards the leaves. Each node in the concept tree describes a class of instances and hence each cut of the concept tree represents a partition of all the known instances.

Given a new instance, the task of the Cobweb algorithm is to create a new concept hierarchy that incorporates the knowledge embodied in the instance. The classification of the new instance proceeds recursively through each level of the concept hierarchy, following a path composed of increasingly specific concept nodes. At each level in the concept hierarchy, the classification process is focussed on the immediate subtree (i.e. the children) of the current node and essentially reduces to the best choice among the following four alternatives, namely:

- *Incorporate* the instance into an existing concept. Select the child concept which best fits the new instance to host it.
- *Create* a new child concept to host the new instance.

- *Merge* two children concepts and use the resulting combined concept to host the new instance.
- *Split* a child concept i.e. replace it with its children and then choose the best child to host the instance.

According the Bayesian theory, the best choice among these alternatives is the one that gives the maximal value of the *Bayesian Evaluation Function*, described in the appendixes of this chapter. The algorithm recursively applies the above classification process at each level of the concept hierarchy, incorporating the instance into concepts of increasing specificity. The algorithm halts when it classifies the instance into either:

- A singleton node at the leaf, or
- A new concept at any level in the hierarchy.

In the Knowledge Miner framework, the Bayesian Data Cluster interfaces with the *LDL++* module through the latter's EFI. It receives the data extracted by *LDL++* as input, classifies it and returns the concept tree as output. The module interface provides all the requisite functionality to the external process/user for fine-tuning the classification process.

## 15.5 Discoveries from A Large Knowledge Base

In this and the following two sections, we present the results of three applications of Knowledge Miner to a large common-sense knowledge base called Cyc, a telecommunication database containing a large number of real telephone circuits and a chemical research database representing over 30 years of chemical research results respectively.

Cyc is a large common-sense knowledge base developed at MCC, containing over one million logical assertions (think of them as relations if you like). We applied an earlier version of the Knowledge Miner system (Shen 1992) to two large collections of objects in Cyc: *Person* and *Organization*. At the time, the *Person* collection contained 633 objects and 77 relevant relations, while the *Organization* collection contained 1550 objects and 150 relevant relations.

The metaquery we used in these experiments is as follows:

$P(X\ Y) \wedge Q(Y\ Z) \Rightarrow R(X\ Z)$

where $P, Q$, and $R$ are variables for binary relations, and $X, Y$, and $Z$ are variables for objects.

This metaquery is instantiated on combinations of the 77 relations in the *Person* collection and the 150 relations in the *Organization* collection respectively. From the data in Cyc, 146 patterns were discovered (i.e., were found to have enough supporting data) from the *Person* collection, and 250 from the *Organization* collection. Example patterns discovered from the *Person* collection are as follows:

acquaintedWith(x y)∧languageSpoken(y z)⇒languageSpoken(x z)
studentAtInstitution(x y)∧hasProfessor(y z)⇒likedBy(x z)
primeMinisterOfCountry(x y) ∧ eq(y z) ⇒ headOfGovernmentOf(x z)
computersFamiliarWith(x y)∧languagesThisFamilyOfMachinesCanRun(y z)⇒programsIn(x z)
presidentOfCountry(x y)∧officialLanguage(y z)⇒languageSpoken(x z)
likedBy(x y)∧wearsClothing(y z)⇒wearsClothing(x z)
birthDate(y z)∧allSuperAbstrac(x y)⇒birthDate(x z)
laterSubAbstractions(x y) ∧ eq(y z) ⇒ startsBeforeStartingOf(x z)
children(x y) ∧ eq(y z) ⇒ relative(x z)
duringActorIn(x y) ∧ eq(y z) ⇒ canPerform(x z)
isAwareOf(x y) ∧ allSubAbstrac(y z) ⇒ isAwareOf(x z)
residences(x y)∧officialLanguage(y EnglishLanguage)⇒speaks(x EnglishLanguage)
memberOfPoliticalParty(x y)∧allInstances(y CommunistParty)⇒ideology(x Communist)
staffMembers(x y)∧eq(y MCC)⇒probableExpertiseInComputers(x UserLevel)

All of these assertions were new to Cyc (i.e, cannot be inferred by Cyc itself at the time). Most of these assertions are interesting because they specify things that are true with high, but less than one, probabilities. These rules can be used by Cyc as default rules for reasoning. For example, the pattern studentAtInstitution(x y)∧hasProfessor(y z)⇒likedBy(x z) says that if $x$ is a student in an institution, then $x$ must be liked by all the professors in that institution. This is not always true but is a good default rule. Given the fact that more and more knowledge bases are capable of reasoning nonmonotonically, this kind of default knowledge is important if you don't know whether a particular student $x$ is liked by a particular professor $z$.

## 15.6 Discoveries from A Telecommunication Database

We also applied KM to a large telecommunication database. This database contains approximately 300MB of information about existing telephone circuits in a regional Bell company. There are two types of circuits in the database: point to point circuits, and multiple-leg circuits. Each circuit (or leg) is a sequence of connected components such as *channel, cable, station, equipment, test point*, etc.. Each component is associated with a set of features. For example, an *equipment* has location, function, channelID, and others. A *channel* has starting and ending locations, conductivity, and others.

We were interested in finding the feature correlation between components that are directly connected in some circuit. So the metaquery used was as follows:

$$Linked(X,Y) \wedge Feature1(X) \wedge Feature2(Y) \Rightarrow equal(X,Y)$$

where $X$ and $Y$ are variables for components. From this metaquery, the instantiator generated a series of *LDL++* queries, with *Linked* bound to a circuit connection table, and *Feature1* and *Feature2* bound to various columns in each component table. Running these *LDL++* queries against the database, the KM system generated 281 patterns in about 7 hours. Two examples of patterns that have high probabilities are:

(P=1.0) linked(EQPT,EQPT)⇒equal(chanID(EQPT),chanID(EQPT))
(P=0.98) linked(CHANNEL,EQPT)⇒equal(endLocation(CHANNEL),location(EQPT))

Two examples of patterns that have low probabilities are:

(P=0.015) linked(LOOP,CABLE)⇒equal(startLocation(LOOP),endLocation(CABLE))
(P=0.004) linked(EQPT,CHANNEL)⇒equal(calcSigVoice(EQPT),ovrdSigVoice(CHANNEL))

Two examples of patterns that have middle probabilities are:

(P=0.59) linked(TPoint,TPoint)⇒equal(accsys(TPoint),accsys(TPoint))
(P=0.40) linked(EQPT,TPoint)⇒equal(hierchyIND(EQPT),analRingSigDir(TPoint))

We presented these rules to experts in this field and most of the rules were deemed to make perfect sense. There are a number of ways that these rules can be used: Those with high probabilities can run directly (they are in fact *LDL++* queries) against the database to find the set of data that violate the rules. Such data is very likely to be erroneous. Similarly, one can run the low probability rules to find the data that obeys the rules - this data is also very likely to be erroneous. These rules can also be used as data validation filters for future data entry. Any input data that violates the high-probability rules (or obeys the low-probability rules) should be flagged and checked before actually being entered into the database. The rules with middle probabilities are also interesting and useful in that they identify subsets of the database that need further investigation. One can then introduce new metaqueries that are constrained to focus on these subsets and subsequently generate new rules with high and low probabilities.

## 15.7 Discoveries from A Chemical Research Database

We are currently applying the KM system to a real chemical research database. This database contains results representing over 30 years of chemical research and experiments. Conceptually speaking, it has information about the ingredients and the properties of various compounds. The ingredients are specified in a single table, while the properties are distributed over more than 50 tables. All these tables are linked via the compound identification.

We are interested in discovering the relationship between ingredients and properties, so the following metaquery is employed:

$$Ingredient(C\ X\ Y) \wedge Property(C\ Z) \Rightarrow Cluster(Z)$$

where $C$ is a variable for compound identification, $X$ and $Y$ are variables for ingredients, and $Z$ is a variable for property. As one may expect given the large number of properties in the database, this metaquery is too time consuming to run. So we restrict it by constraining the variables $X$ and $Y$ to be bound only to those ingredients that are commonly used.

Using the restricted metaquery, KM returns a set of patterns describing the relationships between specific ingredients and properties. One example of these patterns is the relationship between an ingredient *3060* and the *densityBM* property of the compound:

ingredient(C,3060,$X$)$\wedge$ densityBM(Y) $\Rightarrow$ Clusters(Y)=$\{$(1.243,0.46,0.99), (-999.9,0.2,0.01)$\}$

The rule indicates that for this type of compounds, the majority (99%) of values of property densityBM are near 1.243 (with a variance 0.46). Thus, the ingredient 3060 has a very specific effect on the compound's densityBM property, irrespective of the other ingredient $X$. This information is extremely valuable to the chemists. This rule also indicates (with a very low probability cluster) that there may be some noisy data in the densityBM table.

We are currently experimenting with various metaqueries, and the results will be reported in the near future.

## 15.8 Generating Initial Metaqueries

As we have seen so far, metaqueries play an important role in the KM discovery system. However, good metaqueries do not come easy. They must balance the computational expense on one hand with the potential profit of discovery on the other. One may have an extremely general metaquery for a database, but it's demands on computational resources may be untenable. On the other hand, one may have an extremely specific metaquery with fast response characteristics, but it may not discover anything interesting. The ideal case would be to have a metaquery general enough to support the discovery of interesting patterns, but constrained enough to do so without making unreasonable demands on computational and time resources.

Obviously, designing the right metaquery is a difficult problem - in addition to knowledge about the database schema, one also needs good judgement and a healthy dose of intuition. We do not claim to have reduced this process to an exact science - indeed, we have not solved this problem. Instead, our philosophy, as reflected in the KM methodology, is to play the role of an informed assistant and afford a human expert the fullest flexibility and help in defining the best metaqueries he or she can. For example, one thing we find useful is to have the system automatically generate a set of initial metaqueries, which can then be analyzed and refined by the human expert. As more rules are discovered from the database, the system should be able to generate more metaqueries in an interactive fashion. This is the task of the Metaquery Generator in KM.

The Metaquery Generator accepts two inputs: the database schema and the rules discovered from the database. Initially, the generator examines the database schema and generates simple metaqueries that address the relationships across tables. Certain types of information from the schema can provide useful guidance in narrowing down the set of possible metaqueries. These include:

1. Keys
2. Column Types
3. Column Names

Some example heuristics based on these informaton types include:

- Since the key column of a table is always unique, we can pretty much eliminate it from consideration as a possible candidate for instantiation in metaquery.
- When joining any two tables, we need to determine the join columns i.e. a column from each table which could be deemed equal. In this case, the basic criterion is that both columns must have the same types to even have a reasonable chance at success.
- If the column names are the same, the likelihood is greater that the two tables should be joined based on those two columns.

Discovered rules can also be used to guide the generation of metaqueries. For example, knowing the probabilistic distribution of the candidate columns for joining two tables, we can determine whether a metaquery based on the same will generate useful results or not. Thus, less useful metaqueries will not be suggested to the users. Another way to generate metaqueries is to generalize the existing rules. For example, the system BLIP (Theime 1989) uses the heuristic of "turning constants to variables" to acquire metalevel rules from domain-level rules entered by users.

The Metaquery Generator is by no means complete. We have yet to generate complex queries automatically. There is also potentially a lot more useful information that can be gleaned from the database schema and the discovered rules that could be gainfully applied to the generation process. This is ongoing work and we are engaged in a little discovery activity ourselves - we will have more to report as more experiements are carried out.

## 15.9 Related Work

The number of knowledge mining systems that uses artificial intelligence techniques are growing but few combines deductive techniques with inductive techniques. (Matheus, Chan and Piatetsky-Shapiro 1992) compares in detail the methodologies between the various systems.

A comparison framework was suggested by (Simoudis, Livezey and Kerber 1994) based on two aspects. The first is based on the operations supported and the second is based the degree of integration. The list of operations are:

- testing and refinement of rules hypothesized
- automatic discovery of rules by the system
- integration of the best of the hypothesized and discovered rules into a cohesive model

which are all supported by the `Recon` (Simoudis, Livezey and Kerber 1994) and the `IMACS` (Brachman, Selfridge, Terveen, Altman, Halper, Kirk, Lazar, McGuiness, Resnick. and Borgida 1993) system. All these three operations are also supported by our *metaqueries* approach. Other systems such as `KDW` (Piatetsky-Shapiro and Matheus 1992), `INLEN` (Kaufman, Michalski and Kerschberg 1991), `MLT` (Usznski 1992) support only a subset of these operations.

One significant difference between `IMACS` and the `Recon` or our *metaqueries* approach is that the former loads all data into memory from databases for processing while the latter two systems interface database directly and retrieve data as needed at processing time. All three systems are supported by some form of deductive engine. However, since `Recon` can only support non-recursive queries and thus, limited by the SQL it generates, `IMACS` can express and process more complex structures. Our metaqueries approach, which uses the *LDL++* deductive database engine that can support a full range of both recursive and non-recursive queries, has similar expressive and processing power as `IMACS`.

In terms of integration, `Recon`, `KDW` and our *metaqueries* uses a loose architecture where the various components are separated into distinct modules and they interface through well-defined interfaces. In the *metaqueries* approach, second-order predicates or templates are used to interface with the users or the visualization module, first-order predicates are used to interface with the deductive database engine, SQL statements are used to interface with the underlying databases, first-order predicates are used to interface with the clustering routines and rules are generated as input to the knowledge base. `IMACS`, `INLEN` and `MLT` are in comparison more tightly integrated systems.

## 15.10 Conclusions and Future Work

We have presented a framework for using metaqueries to integrate induction and deduction for knowledge discovery from databases. The framework has been implemented in the Knowledge Miner system to integrate two very advanced deductive and inductive technologies: the *LDL++* system and the Bayesian Data Cluster. Applications of Knowledge Miner to several real-world databases and a knowledge base have demonstrated that the metaquery approach is indeed viable and shows much promise.

Our experiments with the system thus far have been fruitful, but a lot more work needs to be done before the KM system can be truly useful in a variety of discovery environments. On the theoretical side, we are exploring more advanced learning algorithms with a better, more intelligent clustering capability as well as a higher efficiency in terms of computational time and space used.

On the practical side too, there are a number of issues on the agenda. One is to have the ability to connect with legacy databases provided by various vendors. The *LDL++* system can currently access data from Sybase, Ingres, Rdb and Oracle. We plan to extend it to also access other database products such as IBM/DB2 and IBM/IMS. A second issue is the design and implementation of an appropriate visualization tool for the graphical presentation of the results of the discovery process.

Lastly, we will also explore new uses for the discovered rules and statistical data. In addition to using them to enforce integrity constraints for future updates and for generating new metaqueries, we believe that they can also be effectively used for estimating the cost of queries and also for supporting some forms of fuzzy querying. These beliefs remain to be verified.

## Bibliography

Apt, Krzysztof R., *Logic Programming*, Chapter 10, Handbook of Theoretical Computer Science, Ed. J. van Leeuwen, pp. 493-574, 1990.

Arni, Ong, Tsur, and Zaniolo. 1994. *LDL++*: A Second Generation Deductive Database System, Working Paper.

Brachman R., Selfridge P., TerveenL., Altman B., Halper F., Kirk T., Lazar T., McGuiness D., Resnick L. and Borgida A., 1993. Integrated support for data archaelogy, *Proceedings 1993 AAAI Workshop on Knowledge Discovery in Databases*, pages 197-211, AAAI.

Cercone, N. and M. Tsuchiya. Guest editors. 1993. Special issue on Learning and Discovery in Databases. *IEEE Transactions on Knowledge and Data Engineering*, 5(6).

Fisher, D., 1987. Knowledge acquisition via incremental conceptual clustering, *Machine Learning*, 2:139-172.

Jaynes, E. 1995. *Probability Theory — The Logic of Science*, Combridge University Press (to appear).

Kaufman K., Michalski R. and Kerschberg L., Mining for knowledge in databases: Goals and general description of the INLEN system, *Proceedings of the 1991 AAAI Workshop on Knowledge Discovery in Databases*, pages 35-51, AAAI 1991.

Kerber, R. and B. Livezey and E. Simoudis. 1994. Recon: A Framework for Database Mining. *International Journal of Intelligent Information Systems.*

Lloyd, J., 1984. *Foundations of Logic Programming*, Springer-Verlag, Chapter 1.

Matheus C., Chan P.K. and Piatetsky-Shapiro G., 1993. Systems for knowledge discovery in databases, *IEEE Transactions on Knowledge and Data Engineering*, 5(6).

Naqvi and Tsur. 1989. *A Logical Language for Data and Knowledge Bases.* W.H. Freeman Company.

Piatetsky-Shapiro G. and Matheus C., 1992. Knowledge discovery workbench for exploring business databases, *International Journal of Intelligent Systems*, 7.

Piatetsky-Shapiro, G. Editor. 1993. Proceedings of AAAI-93 Workshop on Knowledge Discovery in Databases. AAAI Press.

Shen, W.-M. 1992. Discovering Regularities from Knowledge Bases. *International Journal of Intelligent Systems*, 7(7), 623-636. Special issue on Knowledge Discovery in Databases and KnowledgeBases (Selection of best papers from KDD-91 workshop). G. Piatetsky-Shapiro, guest editor.

Shen, W.-M. 1994. *Autonomous Learning from the Environment.* Computer Science Press. W.H. Freeman.

Simoudis E., Livezey B. and Kerber R., 1994. Integrating Inductive and Deductive Reasoning for Database Mining *Proceedings of the 1994 AAAI Workshop on Knowledge Discovery in Databases*, pages 37-48, AAAI.

Theime, S. 1989. The Acquisition of Model-Knowledge for A Model-Driven Machine Learning Approach. In Morik, K. ed. *Knowledge Representation and Organization in Machine Learning.* Springer-Verlag.

Usznski M., 1992. Machine learning toolbox, Technical report, *European Economic Community, Espirit II.*

Zaniolo, C. 1992. Intelligent Databases: Old Challenges and New Opportunities. *Journal of Intelligent Information Systems*, 1. 271-292.

## Appendix 15.A: The Bayesian Evaluation Function

An instance is described by its attributes. Thus, if $K$ is the number of attributes, an instance $D$ can be defined as a vector of $K$ values for these attributes:

$$D \equiv (x_1 = v_1, ..., x_k = v_k, ..., x_K = v_K)$$

A concept is a collection of related or similar instances. The size of a concept $C$, denoted as $|C|$, is the number of its instances. The concept $C$ can be defined as a vector of $K$ probability distribution functions on the $K$ attributes of its instances:

$$C \equiv (f_1, ..., f_k, ..., f_K).$$

where $f_k$ is determined by the $k$th values of all its instances. For instance, if $v_{k1}$, $v_{k2}, \ldots, v_{kN}$ are the $k$th values of $N$ instances $D_1, D_2, \ldots, D_N$, respectively, then, assuming a normal distribution, the probability distribution function $f_k$ can be estimated as:

$$f_k(x_k|D_1 \cdots D_N I) = M_k \exp\left\{-\frac{(x_k - a_k)^2}{2\sigma_k^2}\right\} \tag{15.10.1}$$

where $I$ is the background information

$$M_k = \frac{1}{\sigma_k\sqrt{2\pi}},$$

is a normalization constant to satisfy the constraint:

$$\int_{-\infty}^{+\infty} f_k(x_k|D_1 \cdots D_N I) dx_k = 1,$$

$a_k$, the mean of $f_k$, can be estimated as:

$$a_k = \frac{1}{N}\sum_{j=1}^{N} v_{kj} \tag{15.10.2}$$

and $\sigma_k^2$, the variance of $f_k$, can be estimated as:

$$\sigma_k^2 = \frac{1}{N}\sum_{j=1}^{N}(v_{kj} - a_k)^2. \tag{15.10.3}$$

The values of $a_k$ and $\sigma_k^2$ can also be estimated incrementally for the $n$th instance based on the values of the same for the $n-1$th instance (see Appendix 15.B). The incremental values are estimated as:

$$a_{kN} = a_{k(N-1)} + \frac{v_{kN} - a_{k(N-1)}}{N} \tag{15.10.4}$$

and

$$\sigma_{kN}^2 = \frac{N-1}{N}\left[\sigma_{k(N-1)}^2 + (a_{k(N-1)} - a_{kN})^2\right] + \frac{(v_{kN} - a_{kN})^2}{N} \tag{15.10.5}$$

Let $H = (C_1, \ldots, C_J)$ represent the children of concept $C$. Let child $C_j$ $(1 \leq j \leq J)$ be the one selected to host new instance $D$, and let $H_j$ represent the new set of children nodes obtained after the definition of $C_j$ changes to incorporate $D$. Then how good the assignment of $D$ to $C_j$ is can be estimated by the probability:

$$P(H_j|DH) = P(H_j|H)\frac{P(D|H_jH)}{P(D|H)} \tag{15.10.6}$$

Clearly, the best child concept to host $D$ is $C_i$ such that

$$P(H_i|DH) = \max_{j=1,\ldots,J}\{P(H_j|DH)\} \tag{15.10.7}$$

Assume that $P(H_j|H)$ are equal for all $j$, then to compare $P(H_j|DH)$ we need only compute $P(D|H_jH)$. Notice that

$$P(D|H_jH) = \sum_{j=1}^{J} P(DC_j|H_jH) = \sum_{j=1}^{J} P(C_j|H_jH)P(D|C_jH_jH) \tag{15.10.8}$$

and

$$P(C_j|H_jH) = \frac{|C_j|}{|C|} \tag{15.10.9}$$

and

$$P(D|C_jH_jH) = P(D|C_j) \tag{15.10.10}$$

$P(D|C)$ represents the degree to which instance D belongs to concept $C$ and is computed as:

$$P(D|C) = \prod_{k=1}^{K} P(v_k|f_k) \tag{15.10.11}$$

The term $P(v_k|f_k)$ can be computed easily. If the $k$th attribute is continuous-valud numeric attribute, then

$$P(v_k|f_k) \approx f_k(v_k)\Delta x_k \tag{15.10.12}$$

where $\Delta x_k$ is a small constant range around $v_k$. If the $k$th attribute is discrete (ordered or not), then

$$P(v_k|f_k) = \frac{\text{num. of insts whose } k\text{th value is } v_k}{\text{num. of insts in } C} \tag{15.10.13}$$

## Appendix 15.B: Incremental Update of $a_k$ and $\sigma_k^2$

The parameters of $f_k$ ($a_k$ and $\sigma_k^2$) can be computed incrementally. Let $a_{k(N-1)}$ and $\sigma^2_{k(N-1)}$ be the parameters of $f_k$ after $N-1$ instances. When the $N$th instance is seen, we can use $a_{k(N-1)}$, $\sigma^2_{k(N-1)}$, and $v_{kN}$ (the $k$th value of the $N$th instance) to compute the new parameters $a_{kN}$ and $\sigma^2_{kN}$.

To compute the new mean value $a_{kN}$, notice that

$$\begin{aligned}
a_{kN} &= \frac{1}{N}\sum_{j=1}^{N} v_{kj} \\
&= \frac{1}{N}\sum_{j=1}^{N-1} v_{kj} + \frac{v_{kN}}{N} \\
&= \frac{N-1}{N} a_{k(N-1)} + \frac{v_{kN}}{N} \\
&= a_{k(N-1)} + \frac{v_{kN} - a_{k(N-1)}}{N} \qquad (15.10.14)
\end{aligned}$$

To compute the new variance $\sigma^2_{kN}$, we first observe that

$$\sum_{j=1}^{N-1}(v_{kj} - a_{kN})^2 = \sum_{j=1}^{N-1}(v_{kj} - a_{k(N-1)})^2 + (N-1)(a_{k(N-1)} - a_{kN})^2 \qquad (15.10.15)$$

(see Appendix for the proof of Equation 15.10.15), therefore,

$$\begin{aligned}
\sigma^2_{kN} &= \frac{1}{N}\sum_{j=1}^{N}(v_{kj} - a_{kN})^2 \\
&= \frac{1}{N}\sum_{j=1}^{N-1}(v_{kj} - a_{kN})^2 + \frac{(v_{kN} - a_{kN})^2}{N} \\
&= \frac{1}{N}\left[\sum_{j=1}^{N-1}(v_{kj} - a_{k(N-1)})^2 + (N-1)(a_{k(N-1)} - a_{kN})^2\right] + \frac{(v_{kN} - a_{kN})^2}{N} \\
&= \frac{1}{N}\left[(N-1)\sigma^2_{k(N-1)} + (N-1)(a_{k(N-1)} - a_{kN})^2\right] + \frac{(v_{kN} - a_{kN})^2}{N} \\
&= \frac{N-1}{N}\left[\sigma^2_{k(N-1)} + (a_{k(N-1)} - a_{kN})^2\right] + \frac{(v_{kN} - a_{kN})^2}{N} \qquad (15.10.16)
\end{aligned}$$

These calculations are incremental in the sense that there is no need to refer to any earlier values of instances: $v_{k1}, v_{k2}, \ldots, v_{k(N-1)}$.

Finally, we give the proof of Equation 15.10.15 as follows:

**Proposition:**

$$\sum_{j=1}^{M}(v_j - c)^2 = \sum_{j=1}^{M}(v_j - b)^2 + M(b-c)^2$$

where

$$b = \frac{1}{M}\sum_{j=1}^{M} v_j$$

**Proof:** First, notice that

$$(x-c)^2 = [(x-b)+(b-c)]^2 = (x-b)^2 + 2(x-b)(b-c) + (b-c)^2$$

thus,

$$\begin{aligned}
\sum_{j=1}^{M}(v_j - c)^2 &= (v_1 - b)^2 + 2(v_1 - b)(b-c) + (b-c)^2 + \\
&\quad (v_2 - b)^2 + 2(v_2 - b)(b-c) + (b-c)^2 + \\
&\quad \vdots \\
&\quad (v_M - b)^2 + 2(v_M - b)(b-c) + (b-c)^2 \\
&= \sum_{j=1}^{M}(v_j - b)^2 + 2(b-c)\sum_{j=1}^{M}(v_j - b) + M(b-c)^2 \\
&= \sum_{j=1}^{M}(v_j - b)^2 + 2(b-c)\left[\sum_{j=1}^{M} v_j - Mb\right] + M(b-c)^2 \\
&= \sum_{j=1}^{M}(v_j - b)^2 + 0 + M(b-c)^2 \\
&= \sum_{j=1}^{M}(v_j - b)^2 + M(b-c)^2
\end{aligned}$$

QED.