

Consiglio Nazionale delle Ricerche

Using Deduction for Intelligent Data Analysis

Fosca Giannotti, Giuseppe Manco, Mirco Nanni, Dino Pedreschi,
Franco Turini

Technical Report
CNUCE-B4-1999-02

CNUCE

Pisa

Using Deduction for Intelligent Data Analysis

Fosca Giannotti, Giuseppe Manco

CNUCE - CNR

Via S. Maria 36, I-56126 Pisa, Italy

F.Giannotti@cnuce.cnr.it, G.Manco@cnuce.cnr.it

Mirco Nanni, Dino Pedreschi, Franco Turini

Dipartimento di Informatica, Università di Pisa

Corso Italia 40, I-56125 Pisa, Italy

mnanni@di.unipi.it, pedre@di.unipi.it, turini@di.unipi.it

January 25, 1999

Abstract

A Logic-based query language with built-in data mining primitives is presented. The aim is to have a language where knowledge extraction and knowledge manipulation are integrated. Its flexibility and expressiveness in supporting the process of knowledge discovery in databases is illustrated by presenting a market-basket analysis application. By focussing on association rules, we show that the query language deals effectively and uniformly with data preparation, rule extraction, analysis and construction of business rules. The implementation of the language is based on an architecture that allows a convenient compromise between tight and loose coupling of database querying and data mining.

1 INTRODUCTION

Motivation. The process of making decisions requires the combination of two kind of activities: knowledge acquisition and reasoning on the acquired knowledge according to the expert rules that characterize the business.

Data mining techniques are an answer to the first issue in that they extract from raw data knowledge that is implicit and, more important, that is at a higher abstraction level. In fact, as massive amounts of data become available everyday from every kind of organizations, the ability to extract useful knowledge from these data is becoming a crucial need: as the said goes, *we are drowning in data, but starving for knowledge*.

While it is clear which are the various steps of the knowledge discovery process, we are still far away from an integrated methodology and support environment, which makes knowledge extraction feasible. It is now clear that the problem is that each step in the KDD process requires specific tools and expertise, as witnessed by the distance between the data mining tools and the query language. There is a need to move from mining in rough data to mining in databases. More ambitiously, there is a need for combining the inductive capabilities of the data mining tools with reasoning, to the purpose of addressing the difficult analysis tasks posed by key applications.

One such example is market basket analysis, which is rapidly becoming a key factor of success in the highly competing scene of big supermarket retailers. The reason of this fact lies in the consideration that a clear-cut knowledge of customers and their purchasing behavior brings potentially huge added value to retail companies.

Knowledge on purchasing behavior can hardly be discovered using only traditional aggregates or OLAP tools. Any analysis based on aggregated information on sold/delivered goods abstracts away from the essential information contained in each *basket* – the *cash register transaction* that represents the collection of items in a single purchase of a customer. A more effective approach is based on *association rules*, which allow discovering relations between items purchased in the same basket [3, 2, 5]. This form of information is more useful for describing the different purchasing habits of customers. Still, however, association rules are often too low-level to be directly used as a support of marketing decisions. Market analysts expect answers to more general questions, such as “Is supermarket assortment adequate for the company’s target class of customers?” “Is a promotional campaign effective in establishing a desired purchasing habit in the target class of customers?” These are *business rules*, and association rules are necessary, albeit insufficient, basic mechanisms for their construction. Business rules require also the ability of combining association rule mining with *deduction*, or *reasoning*: reasoning on the temporal dimension, reasoning at different levels of granularity of products, reasoning on the spatial dimension, reasoning on association rules themselves.

Objectives. The objective of this paper is precisely to demonstrate how a suitable integration of deductive reasoning, such as that supported by logic database languages, and inductive reasoning, such as that supported by association rules, provides a viable solution to many high-level problems in many applications where a support to decision making is needed. Put another way, we believe that the key to succeed in constructing effective decision support systems is the ability of integrating the results of knowledge extraction with expert rules. We then maintain that a logic database framework is the right basis for constructing such integration.

We briefly present Datasift, a prototype system that integrates the deductive capabilities of a logic-based database language, LDL++ [1], with the inductive capabilities of diverse data mining algorithms and tools, notably association rules. The effectiveness of the proposed system in dealing with knowledge extraction in general, and with the various aspects of market basket analysis in particular, is discussed through a series of examples, covering data preparation, rule extraction, post-processing, and business rules. The application has been developed in collaboration with COOP Toscana Lazio, a division of one of the major Italian supermarket companies.

The approach described in this paper shares motivations and ideas with other proposals, which are later discussed in section 4. However, the novelty of the present proposal is twofold. At the level of the query language, the proposed fine-grained integration of induction and deduction supports a higher degree of expressiveness, which revealed essential in tackling the market basket application. At the level of system architecture, we propose a form of

coupling of the query language and the mining tools which is a compromise between loose and tight coupling, aimed at minimizing the drawbacks of the two approaches and combining their advantages.

Plan of the paper. The next section illustrates the logic-based query language and its data mining extensions. Section 3 is devoted to put the language at work in the market basket application, while Section 4 discusses the comparisons with relevant related work. Finally, Section 5 sketches the architecture of our system, and some concluding remarks are presented in Section 6.

2 A LOGIC QUERY LANGUAGE WITH DATA MINING MECHANISMS

Deductive databases are database management systems whose query languages and storage structures are designed around a logical model of data. The underlying technology is an extension to relational databases that increases the power of the query language. Among the other features, the rule-based extensions support the specification of queries using recursion and negation.

LDL++: A logic query language. We adopt the LDL++ deductive database system, which provides, in addition to the typical deductive features, a highly expressive query language with advanced mechanisms for non-deterministic, non-monotonic and temporal reasoning [6,11].

In deductive databases, the extension of a relation is viewed as a set of facts, where each fact corresponds to a tuple. For example, let us consider the predicate `assembly(Part,Subpart)` containing parts and their immediate subparts. The predicate `part_cost(BasicPart,Supplier,Cost)` describes the basic parts, i.e., parts bought from external suppliers rather than assembled internally. Moreover, for each part the predicate describes the supplier, and for each supplier the price charged for it.

```
assembly(bike,frame).      part_cost(top_tube,reed,20).
assembly(bike,wheel).     part_cost(fork,smith,10).
assembly(frame,top_tube). part_cost(top_tube,smith,25).
assembly(frame,fork).     part_cost(spoke,reed,5).
assembly(wheel,spoke).    part_cost(nipple,smith,3).
assembly(wheel,nipple).
```

Rules constitute the main construct of LDL++ programs. For instance, the rule

```
multiple_supp(S) ← part_cost(P1,S,_) , part_cost(P2,S,_) , P1 ≠ P2.
```

describes suppliers that sell more than one part. The rule corresponds to the SQL join query

```
SELECT P1.Supplier
FROM part_cost P1, part_cost P2
WHERE P1.Supplier = P2.Supplier
      AND P1.BasicPart ≠ P2.BasicPart
```

In addition to the standard relational features, LDL++ provides recursion and negation. For example, the rule

```
all_subparts(P,S) ← assembly(P,S).
all_subparts(P,S) ← all_subparts(P,S1) , assembly(S1,S2).
```

computes the transitive closure of the relation `assembly`. The following rule computes the least cost for each basic part by exploiting negation:

```

cheapest(P,C) ← part_cost(P,_,C), ¬ cheaper(P,C).
cheaper(P,C) ← part_cost(P,C1), C < C1.

```

A remarkable capability is that of expressing *distributive aggregates* (i.e., aggregates computable by means of a distributive and associative operator), which are definable by the user [15]. For example, the following rule illustrates the use of a sum aggregate, which aggregates the values of the relation `sales` along the dimension `Dealer`:

```

supplier_tot(Date, Place, sum<Sales>) ←
    sales(Date,Place,Dealer,Sales).

```

Such rule corresponds to the SQL statement

```

SELECT Date, Place, SUM(Sales)
FROM sales
GROUP BY Date, Place

```

Starting from tuples such as

```

sales(2/2/1994, new york, smith, 10000).
sales(2/2/1994, new york, reed, 15000).

```

the rule produces tuples such as `supplier_tot(2/2/1994, new york, 25000)`.

The typical functionalities provided by any OLAP system include the organization of data in a multidimensional database, i.e. a data-cube that enables us to perform analysis and to aggregate data along its dimensions. A simple way of dealing with multidimensional views in LDL++ exploits aggregation and recursion. The various aggregations can be obtained in fact by stratifying the construction among the various dimensions:

```

cuboid(0,Date,Place,Sale) ←
    sales(Date,Place,Dealer,Sale).

cuboid(I+1, Da, Db,sum<S>) ← cuboid(I, D1, D2,S),
    path(D1, D2, Da, Db).

path(Date, Place,Date,*) ← Place ≠ *, Date ≠ *.
path(Date, Place,*, Place) ← Place ≠ *, Date ≠ *.
path(*,Place,*,*) ← Place ≠ *.

cube(D1, D2,S) ← cuboid(_,D1, D2,S).

```

Here, `path` computes an aggregation pattern on the basis of the last computed patterns. Further functionalities, such as rollup, slicing and pivoting, can be easily obtained by combining again recursion and aggregates.

It is worth noting how the SQL formalism is not powerful enough to take advantage of the distributive nature of the sum aggregation. In fact, the data cube sum of the sales can be computed in SQL with the following expression:

```

SELECT '*', '*', SUM(Sales)
FROM sales
UNION
SELECT Place, '*', SUM(Sales)
FROM sales
GROUP BY Place

```

```

UNION
  SELECT '*', Year, SUM(Sales)
  FROM sales
  GROUP BY Year
UNION
  SELECT Place, Year, SUM(Sales)
  FROM sales
  GROUP BY Place, Year

```

that corresponds to the enumeration of the possible aggregates that need to be computed. This union query is however extremely inefficient, in that it recomputes from scratch along each dimension of the cube.

Adding a mining mechanism. A general way of dealing with data mining in a deductive framework is to directly define queries, which implement such mining needs. Aggregates are basic tool to start with. For example, the following program defines typical (two-dimensional) association rules by using the predefined count aggregate.

```

pair(I1, I2, count<T>) ← basket(T, I1), basket(T, I2).
rules(I1, I2) ← pair(I1, I2, C), C ≥ 2.

```

The first rule generates and counts all the possible pairs, and the second one selects the pairs with sufficient support (i.e., at least 2). As a result, predicate `rules` specifies associations, i.e. rules that state that certain combinations of values occur with other combinations of values with a certain frequency. Given the following definitions of `basket`,

```

basket(1, fish).      basket(3, bread).
basket(1, bread).    basket(3, orange).
                    basket(3, milk).

basket(2, bread).
basket(2, milk).
basket(2, onions).
basket(2, fish).

```

by querying `rules(X, Y)` we obtain the answers `rules(milk, bread)`, `rules(bread, milk)` and `rules(fish, bread)` and `rules(bread, fish)`.

Using aggregates it is easy to define concepts of interestingness of the rules diverse from the usual statistical parameters. Thus, other interesting measures, such as financial measures, may be defined. For example, if we are interested in discovering the associations between the cities and the products where the sales decreased of more than 30% w.r.t. the average, we can define the following query:

```

average(avg<Sales>) ← sales(City, Product, Date, Sales).

avg_cp(City, Product, avg<Sales>) ← sales(City, Product, Date, Sales).

answer(City, Product) ← average(A), avg_cp(City, Product, P), P ≥ 0.70 × A.

```

The first rule computes the average on the whole sales. The second rule computes the averages related to the tuples `<City, Product>`, and the third rule selects the relevant rules.

However, the idea of using the query language to directly implement mining algorithms is not a novelty and it raises obvious concerns about efficiency. In our proposal, we decided to use aggregates as the means to introduce mining primitives into the query language, and to implement such aggregates by exploiting another characteristics of the LDL++ system,

namely, its open architecture, which supports easy connectivity with a variety of external systems and components.

Association rules. Let I be a set of data items, and T a set of tuples with values ranging over I . The problem of discovering association rules can be defined as finding relationships between the occurrences of subsets of I within tuples of T [5,7]. An association rule of form $X \Rightarrow Y$ is described in terms of support and confidence. The support of X over the set of tuples of T is the fraction of the tuples that contain X . The confidence of a rule $X \Rightarrow Y$ over a set of tuples T is the fraction of tuples containing X that also contains Y .

In our language, association rules are computed by an aggregate, as illustrated in the following rule:

```
rules(patterns<(min_supp, min_conf, Y1, ..., Yn)>) ← q(X1, ..., Xm).
```

In this rule, the variables Y_1, \dots, Y_n are a subset of the variables X_1, \dots, X_m of q . The aggregate `patterns` computes the set of quadruples (L, R, S, C) where:

1. L and R are respectively the left and right side of an association rule $L \Rightarrow R$,
2. $L = \langle l_1, \dots, l_1 \rangle$ and $R = \langle r_1, \dots, r_k \rangle$ where the tuple $\langle l_1, \dots, l_1, r_1, \dots, r_k \rangle$ is a rearranged subset of the values of Y_1, \dots, Y_n in a tuple resulting from the evaluation of q .
3. S, C are respectively the (normalized) support and confidence of the rule $L \Rightarrow R$, such that $S \geq \text{min_supp}$ and $C \geq \text{min_conf}$.

As an example, the following program computes the two-dimensional association rules with a minimum 40% support:

```
rules(patterns<0.4, 0, I1, I2>) ← basket(T, I1), basket(T, I2).
```

Here, by querying `rules(L, R, S, C)`, we obtain the following tuples:

```
rules(milk, bread, 0.66, 1)      rules(bread, milk, 0.66, 0.66)
rules(fish, bread, 0.66, 1)     rules(bread, fish, 0.66, 0.66)
```

Here an external ad hoc induction algorithm computes the patterns, while the deductive engine has the only task of exchanging the data to the inductive engine on demand.

With the same strategy, other mining mechanisms have been integrated in the language, including a form of bayesian clustering computed with the Autoclass algorithm, and a classification tool based on C4.5. It is currently under investigation the extension with temporal series. The discussion on such extensions is out of the scope of the paper which concentrates on the use of association rule mining for market basket analysis.

3 A MARKET BASKET ANALYSIS APPLICATION

As an example study of the modeling capabilities of the proposed approach, we now instantiate the proposed deductive/inductive system in the specific domain of market basket analysis. After discussing the database schema, we concentrate on the data preparation, or pre-processing, phase, the rule extraction phase, and the post-processing phase. We finally see how these phases can be combined to perform more complex analysis, which we call *business rules*.

Database schema. Figure 1 illustrates the entities involved in the domain of market basket analysis. The main entity is the cash-register transaction, corresponding to a basket of items (products) purchased by a customer at a given time in a given location (store). Time, location and, if available, purchaser are dimensions which allow multiple granularities and aggregations. Analogously, items are also organized into hierarchies: single products are grouped into families, families are grouped into sectors, sectors are grouped into departments, and so on.

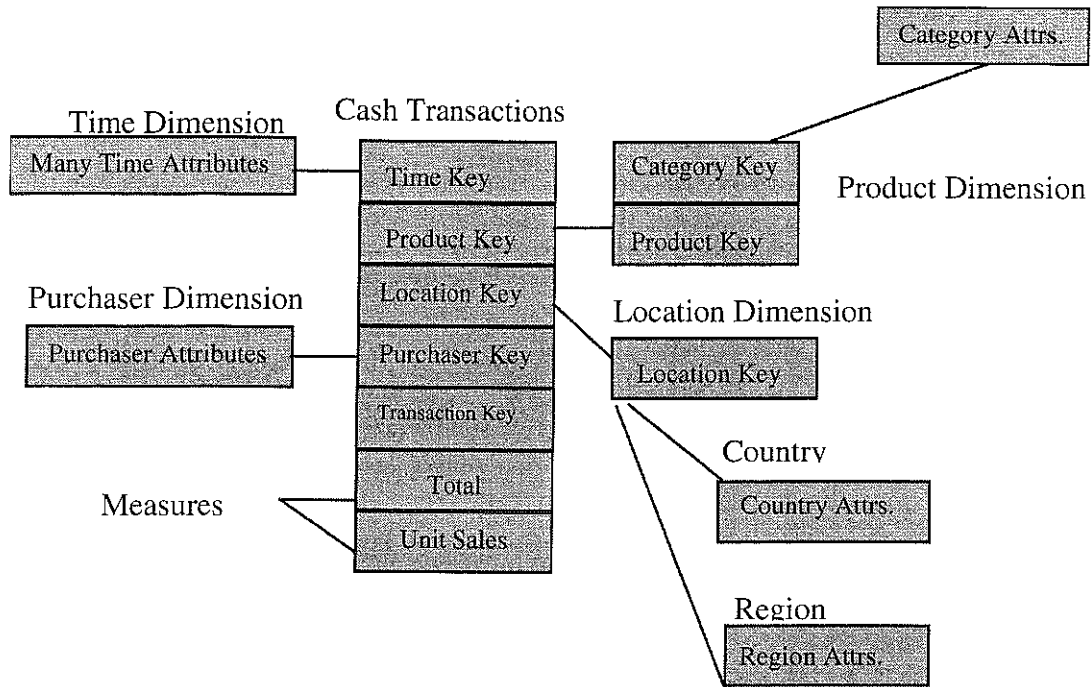


Figure 1: database schema.

Data preparation. By exploiting the deductive framework, we can easily express many useful queries that partition and group data in order to focus the analysis on a particular subset and/or a particular view:

- *Selection w.r.t. values/dimensions.*

We can partition data in segments, according to some specified values or in order to enhance/hide some particular properties of a specific dimension. For example the following rules specify a partition w.r.t. a time ontology (where the interval facts specify a predefined granularity we are interested to).

```
interval(label1, 2/2/1998, 2/16/1998).
.
.
.
interval(labeln, 4/8/1998, 4/18/1998).
```

```
partition(Label, Tid, Item) ←
    cash_transaction(Tid, Date, ..., Item),
    interval(Label, Start, End),
    Start ≤ Date, Date ≤ End.
```

Analogously, we can cut off from the transactions a set of uninteresting items specified by some constraints (e.g., $Item \neq \text{plastic bag}$).

- *Hierarchies.*

Usually, single items in a retail context are grouped into categories, that in turn can be grouped into categories, thus forming a concept hierarchy. Hierarchies allow the creation of abstraction of items, and consequently of the transactions built on such items. The discovered information can be generalized/instantiated, by navigating along such hierarchy. Additional significant knowledge can be obtained by comparing generalizations with instantiations [9].

An abstraction can be easily modeled in a logic-based language, as the following example shows. Suppose the relation `category(X,Y)` expresses the membership of `X` to category `Y`:

```
category(orange, fruit).
category(banana, fruit).
category(fruit, food).
category(food, all).

items_abstraction(0, Tid, Item) ← basket(Tid, Item).
items_abstraction(I+1, Tid, AbsItem) ←
    items_abstraction(I, Tid, Item),
    category(Item, AbsItem).
```

Now, we can mine knowledge (in our context, association rules) at a given abstraction level `I` by focusing on `items_abstraction(I, ...)`.

Another way of exploiting the existing hierarchy is to focus the analysis to a given category of products. We can obtain this by simply selecting the items that fall in the category, or in a sub-category which in turn falls in the first one. In the following example we define an `is_a` relation as the transitive closure of `category`, and then exploit it in order to focus on the fruit category:

```
is_a(X,Y) ← category(X,Y).
is_a(X,Y) ← category(X,Z), is_a(Z,Y).

focused_dataset(Tid, Item) ← basket(Tid, Item),
    is_a(Item, fruit).
```

Rule extraction. Once the dataset is ready to be mined, we can apply the new mining aggregation operator. We adopt a slight modification of the schema presented above, in order to deal with itemsets of general size:

```
transaction_sets(Id,<Item>) ← basket(Id, Item).
rules(patterns<min_supp, min_conf, Tset>) ←
    transaction_sets(Id, Tset).
```

The first rule collects all the baskets from the `basket` relation. The second rule extracts the relevant rules from the collection of baskets, by applying the induction algorithm.

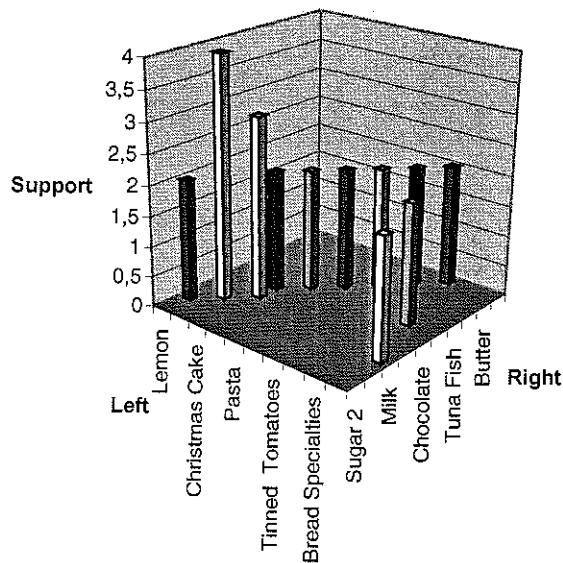


Figure 2: visualization of Association Rules

Post-processing. Since the computed association rules are tuples of a relation, we can manipulate them in the usual ways. We can then, for example, select only the one-to-one rules that involve items from the same category:

```
local_rules(Left, Right, Supp, Conf) ←
    rules({Left}, {Right}, Supp, Conf),
    category(Left, Category), category(Right, Category).
```

Business rules. By a tighter coupling of the data preparation and post-processing steps, we can specify business high-level rules in a straightforward way. We discuss below some interesting examples.

- *Which rules survive/decay up or down the product hierarchy?*
To extract the rules which are preserved in an abstraction step, we can compute rules separately at each abstraction level, and select those which occur both at a level I and at level $I+1$:

```
itemset_abstraction(I, Tid, <Item>) ←
    items_abstraction(I, Tid, Item).

rules_at_level(I, pattern<S,C,Itemset>) ←
    itemset_abstraction(I, Tid, Itemset).

preserved_rules(Left, Right) ←
    rules_at_level(I, Left, Right, _, _),
    rules_at_level(I+1, Left, Right, _, _).
```

- *What happens after promoting some product?*
We can give an answer by finding those rules which have been established by the promotion, i.e. rules which did not hold before the promotion, which were raised during the promotion and persisted after the promotion:

```
interval(before, -∞, 3/7/1998).
interval(promotion, 3/8/1998, 3/30/1998).
```

```

interval(after, 3/31/1998, +∞).

itemsets_partition(Label, Tid, <Item>) ←
    partition(Label, Tid, Item).

rules_partition(Label, pattern<S,C, Itemset>) ←
    itemsets_partition(Label, _, Itemset).

preserved_rules(Left, Right) ←
    rules_partition(promotion, Left, Right, _, _),
    ¬rules_partition(before, Left, Right, _, _),
    rules_partition(after, Left, Right, _, _).

```

- *How do rules change along time?*

One way to answer is given by the rules computed separately on each time interval, as explained in the previous subsections. Another, slightly different, consists of computing rules valid in the whole dataset and then checking their support and confidence in the intervals. We then obtain a description of the evolution of rules in time, which can be used for instance to check whether a rule holds uniformly along time or has some peak in an interval, or shows some kind of periodicity.

```

check_support(Set, Label, count<Tid>) ←
    itemsets_partition(Label, Tid, Itemset),
    subset(Set, Itemset).

rules_evolution(Left, Right, Label, Supp, Conf) ←
    rules(Left, Right, _, _),
    union(Left, Right, All),
    check_support(All, Label, Supp),
    check_support(Left, Label, LSupp),
    Conf = Supp/LSupp.

```

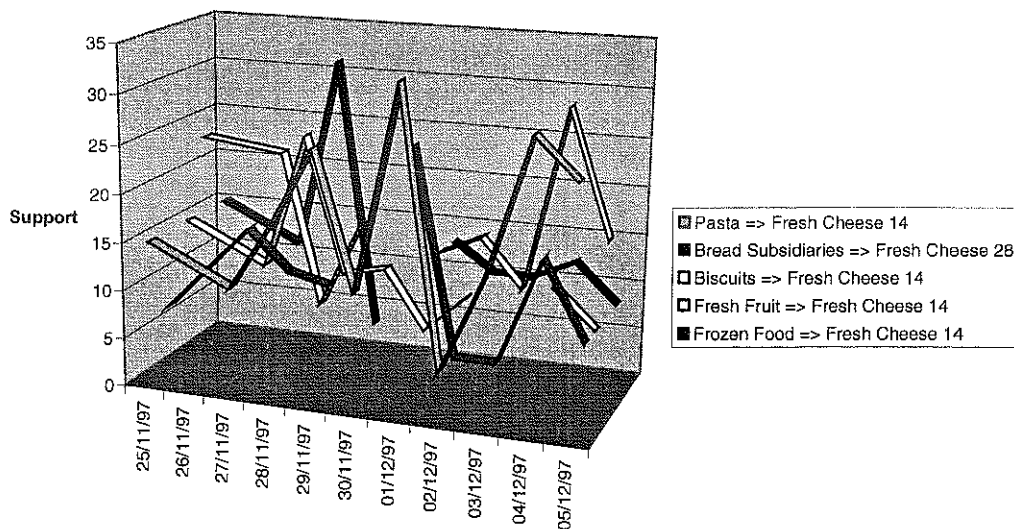


Figure 3: visualization of rules evolution.

- Which rules involve the greatest economical value?

We can compute a *monetary support* for each rule, defining it as the sum of the totals of the transactions to which the rule applies.

```
money_support(Left, Right, sum<Total>) ←
    rules(Left, Right, __, __),
    itemsets(Tid, Itemset),
    union(Left, Right, All),
    subset(All, Itemset),
    cash_transaction(Tid, ..., Total, ...).
```

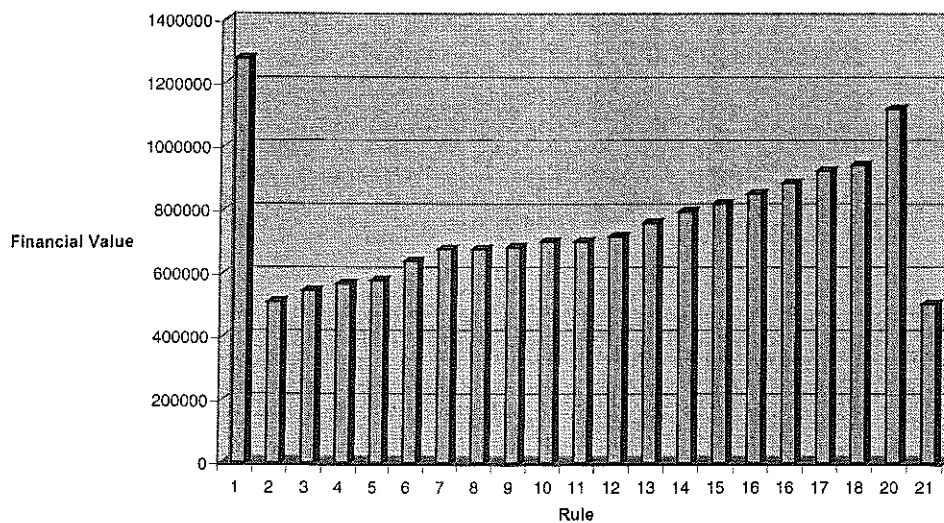


Figure 4: financial support of association rules.

4 COMPARISON WITH OTHER SYSTEMS

The deductive/inductive approach taken in this paper exhibits similarities with the various proposals of integration of data mining with databases by extending SQL to support mining operators. For instance, the query language proposed in [8] extends SQL with the new operator `MINE RULE`, which allows the computation and coding in a relational format of association rules.

As an example, by considering a relation `basket(TransID, CustID, Date, Item, Value)` that contains the transactions of a sales representative, the following rule allows the extraction of the rules with support 20% and confidence 50%:

```
MINE RULE Associations AS
  SELECT DISTINCT 1..n Item AS BODY, 1..1 Item AS HEAD,
  SUPPORT, CONFIDENCE
  WHERE BODY.Value > 100
  AND HEAD.Value > 100
  FROM basket
  GROUP BY CustID
  HAVING COUNT(Item) > 4
  CLUSTER BY Date
  HAVING BODY.Date < HEAD.Date
```

The above expression specifies the mining of associations of purchased items such that the right part of the rule (consisting of only 1 item) has been purchased after the left part of the rule (that can consist of more than one item), and related to those customers who bought more than 4 items. Moreover, we consider only items with a value greater than 100.

Other similar approaches include the DMQL [7] proposal, which extends SQL with a collection of operators for mining different forms of rules. A major problem with such approaches is the limited expressive power of the underlying relational model. Defining complex relations among entities may require set of logical expressions, containing advanced forms of negation and recursion. Clearly, such a limitation is particularly important when the aim is to extract high-level knowledge.

The first proposal of using a deductive database language for explorative data analysis has been developed in [12]. The main idea is that of defining meta-rules that describe patterns. Intuitively, a meta-rule

$$P_1(X_1) \wedge P_2(X_2) \wedge \dots \wedge P_n(X_n) \Rightarrow Q_1(Y_1) \wedge Q_2(Y_2) \wedge \dots \wedge Q_m(Y_m)$$

represents a high-level concept to be verified against the database. By instantiating such concept with rules built on the relations of the databases one can then verify which patterns hold in the database. For example, the meta-rule

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

is analyzed against the database

```

basket(john,sugar,20).
basket(steve,sugar,10).
basket(john,oranges,30).
basket(steve,oranges,40).
basket(john,apples,20).

purchase(X,Y) ← basket(X,Y,_).

avg_value(X,avg<A>) ← basket(X,_,A).

```

by instantiating all the predicate variables, and then executing the right-hand and the left-hand of the obtained instances in order to compare the results. This allows discovering, e.g., the rule

$$\text{purchase}(X, \text{sugar}) \wedge \text{purchase}(X, \text{oranges}) \Rightarrow \text{avg_value}(X, 20)$$

stating that, if a purchaser includes in his basket *oranges* and *sugar*, then usually the average values of his purchased items is 20.

In this approach the logic language is used as a simple computational engine. By using deductions, the system is capable of querying the right and left parts of the (instantiated) meta-rule. However, there's no integration between the deduced knowledge and the induced knowledge, e.g. between the results of the analysis and user defined knowledge.

The query flocks proposal in [10] shows how the combination of association rules with the deductive capabilities of a logic query language yields a higher degree of expressiveness and flexibility, which is crucial in addressing problems such as those posed by market basket

analysis. In this approach the logic language is presented as a good specification formalism while the computational support has to be re-defined for each mining method on the basis of the properties of the problem to be analyzed.

Let us consider the following example:

```
pair(I1, I2, count<T>) ← basket(T, I1), basket(T, I2).
rules(I1, I2) ← pair(I1, I2, C), C ≥ 2.
```

Using the following property of the count aggregate:

```
if count(S) > T then for each S' ⊂ S count(S') > T
```

the program may be transformed into the following one, so yielding to an efficient execution.

```
temp(I, count<T>) ← basket(T, I).
filter(I) ← temp(I, C), C ≥ 2.
pair(I1, I2, count<T>) ← filter(I1), filter(I2),
                             basket(T, I1), basket(T, I2).
rules(I1, I2) ← pair(I1, I2, C), C ≥ 2.
```

The problem of such approach is that it cannot be generalized to other aggregates. For example, the above property does not hold for the average aggregate.

5 OVERVIEW OF DATASIFT ARCHITECTURE

Datasift is a prototype system that implements some features of the model proposed in the previous sections. The Datasift system adopts a hierarchical client-server web-based architecture, in which:

- the client component (*User Interface*) queries the knowledge base (by means of the query engine) and builds the suitable representation metaphors (either graphical or textual);
- a first-level server component (*Query Engine*) maintains the knowledge base, by integrating multiple heterogeneous data sources and by using the other server components to update the local database by means of the available analysis services;
- a second-level server component consisting of the data mining modules and the integrated inductive-deductive query language;
- a third-level server component supporting the access to external databases (DB2 Universal Database Server and Oracle 8 Server).

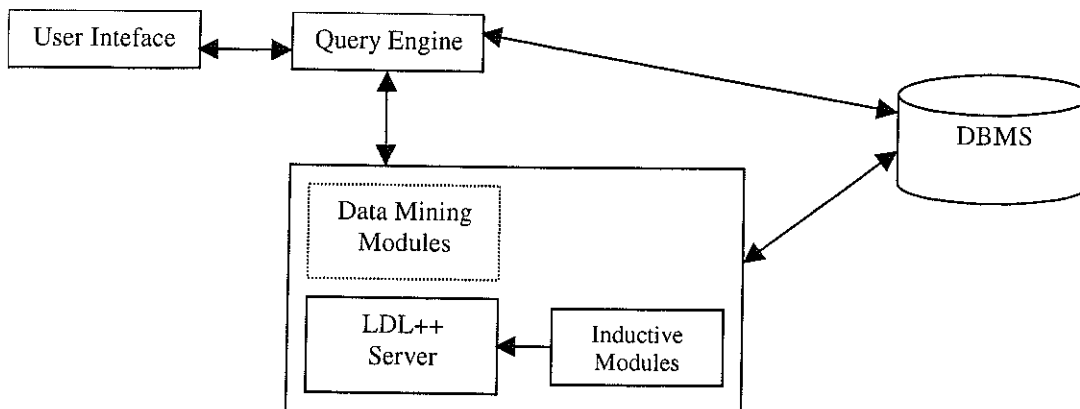


Figure 5: architecture of Datasift.

Client agent. The main role of the client component (*User Interface*) is to build suitable representation metaphors. Two kinds of possible client interactions are provided: through a web browser (that provides a textual representation of the results), and through a MS-Excel application (that is used for both textual and graphical representation). In both cases, the interaction with the server is realized by HTTP connections.

The interaction with the query engine is provided by means of CGI scripts, that provide the requested data. The following classes of possible interactions are supported in the current version of the prototype:

- Extraction of association rules from a database.
- Computation of time series and time evolution of association rules.
- Computation of association rules on the basis of economical relevance.
- Clustering.

We chose to develop a web-based interaction for two main reasons:

1. It is platform-independent, and allows different clients to run on different hosts.
2. It allows the use of standard components, such as the HTTP protocol for the communication between client and server, thus making it easier to develop an interface.

An example snapshot of the current user interface is given in fig. 6. On the left side a list of the available analyses is provided. An example of user interface for the input of parameters for the analysis is shown in fig. 7, which is used in the computation of association rules.

However, a more accurate representation metaphor is currently implemented by means of a MS Excel application that, in addition to the web-based interaction capabilities, allows us to use standard representation metaphors. In this perspective, the Excel application simply downloads the results of the analyses in some pivot tables, and builds standard visualization metaphors on the basis of such tables.

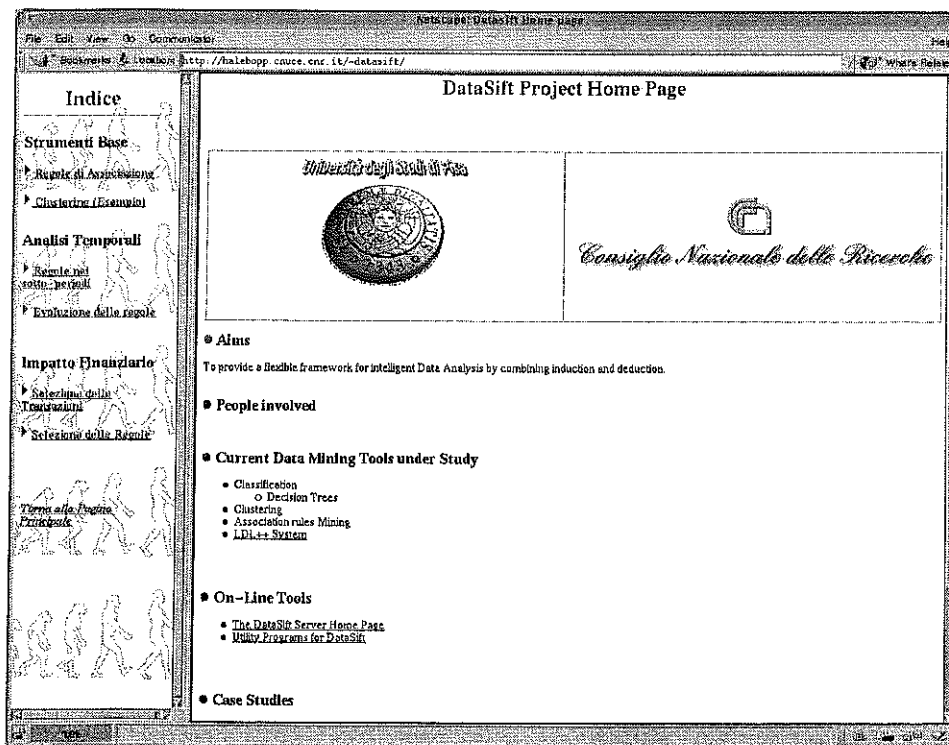


Figure 6: Web interface (main page).

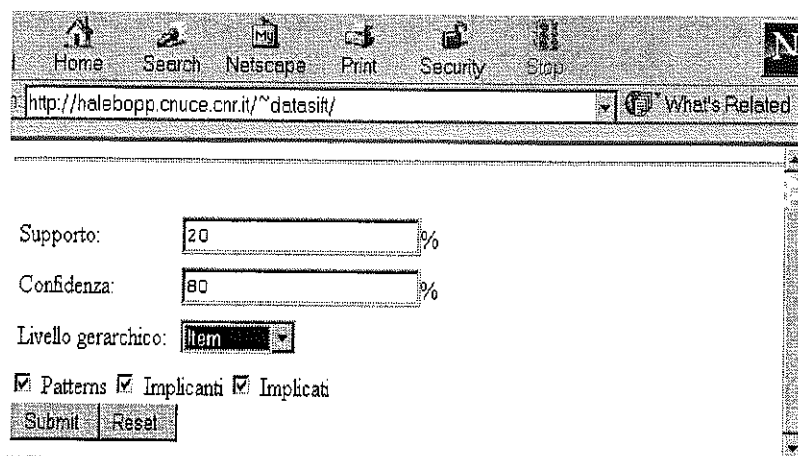
Server agent. The *Query Engine* is an interface to the clients via Web, whose role is to maintain

- a set of relevant statistics over the main database, obtained by directly querying the data, and
- the results of the rule extraction queries committed to the deductive-inductive modules.

The query engine plays a crucial role in the implementation of caching strategies, especially when the analyses are related to massive amount of data. Notice that the maintenance of a local database of results is very important in order to implement incremental mining refinement techniques [4], which is planned as a future extension. In order to populate the database of the current rules, the query engine dispatches to the relative modules the necessary extraction queries. These include specialized data mining algorithms, which do not need preprocessing/refinement steps, and the deductive-inductive module.

The coupling between the core language (LDL++) and the mining tool (association rules) exploits a trade-off between the tight and the loose coupling policies. The LDL++ system provides an open architecture, being able to interface with a variety of external systems and components [1]. This feature reveals very useful for integrating the core language with new functionalities. This allows the definition of user predicates whose evaluation is demanded to external functions.

A user-defined function, representing the new aggregate, can be coded as a C++ routine that efficiently implements special purpose algorithms for solving the given problem, by possibly exploiting any optimized internal data structure. In our system, we implemented an enhanced version of the *Apriori* algorithm for association rules, described in [3]. Moreover, we adapted the *Apriori* algorithm to work directly on the LDL++ objects, i.e. the special data structures that allow to exchange information between the LDL++ system and the C++ routines. This avoids a drawback of loosely coupled systems, i.e. the translation of inputs into flat files before the running of an external special purpose tool and then the backward conversion of outputs, as discussed in [2]. Finally, our solution also avoids the typical drawback of tightly coupled systems, i.e., the lack of efficiency due to inappropriate data structure in that our implementation of *Apriori* algorithm runs fast enough on LDL++ objects.



The screenshot shows a Netscape browser window with the address bar containing "http://halebopp.cnuce.cnr.it/~datasit/". The page content includes the following form elements:

- Supporto: %
- Confidenza: %
- Livello gerarchico:
- Patterns Implicanti Implicati
-

Figure 7: Parameters input interface (association rules page).

6 CONCLUSIONS

In this paper we presented an integrated inductive deductive system for data analysis, and illustrated its functionalities in addressing high-level business rules for market basket analysis. At the present stage, the Datasift system is a prototype, which served as a demonstrator to validate with the end user COOP the viability of the approach. The system revealed an effective tool for the market analysts, who, using the predefined business rules, are able to visualize the answer according to some adequate graphical metaphors, within standard desktop tools. Moreover, we are experiencing that an expert knowledge engineer is generally able to translate quickly the analyst's questions into deductive/inductive rules.

The current version of the Datasift language supports association rules and clustering by means of specialized aggregates; the extension with classification mechanisms and temporal series is currently under development.

A problem in the current prototype is the extensibility of the user interface. While, in fact, it is simple to add new analyses to the deductive-inductive component in a modular way (and to report such additions to the query engine), there is not a uniform way of describing a corresponding representation metaphor. Hence, each time a new analysis (e.g., a business rule) is added, the user interface needs to be consequently extended.

Another main concern is efficiency, a typical problem with deductive database systems, such as LDL++, and a crucial problem when analyzing massive amounts of data. For this reason, in the planned engineering of the Datasift system, we intend to adopt suitable data reduction techniques.

7 ACKNOWLEDGEMENTS

We are indebted with our collaborators at COOP Toscana Lazio for their support during the Datasift project: Massimo Palla, Giuseppe Lallai and Walter Fabbri. Thanks are also owing to Nando Gallo from Intecs Sistemi, Pisa, for his help in designing the visualization component. The project reported in this paper has been supported by Regione Toscana under a grant "Rete Telematica ad Alta Tecnologia".

BIBLIOGRAPHY

1. N.Arne, K.Ong, S.Tsur, C.Zaniolo. LDL++: A Second Generation Deductive Databases Systems. Technical report, MCC Corporation, 1993.
2. R.Agrawal, S.Sarawagi, S.Thomas. Integrating Association Rule Mining with Relational Database Systems: Alternatives and Implications. In Procs. of ACM-SIGMOD'98, 1998.
3. R.Agrawal, R.Srikant. Fast Algorithms for Mining Association Rules. In Procs. of 20th Int'l Conference on Very Large Databases, 1994.
4. E.Baralis, G.Psaila. Incremental Refinement of Association Rule Mining. In Procs. of SEBD'98, 1998.
5. M. J. A. Berry, G.Linoff. Data Mining Techniques for Marketing Sales, and Customer Support. Wiley, 1997.

6. F. Giannotti, G. Manco, M. Nanni, D. Pedreschi. Query Answering in Nondeterministic, Nonmonotonic, Logic Databases. In *Procs. of the Workshop on Flexible Query Answering*, number 1395 in LNAI, 1998.
7. J.Han, Y.Fu, K.Koperski, W.Wang, O.Zaiane. DMQL: A Data Mining Query Language for Relational Databases. In *SIGMOD'96 Workshop on Research Issues on Data Mining and Knowledge Discovery (DMKD'96)*, 1996.
8. R.Meo, G.Psaila, S.Ceri. A Tightly-Coupled Architecture for Data Mining. In *International Conference on Data Engineering (ICDE98)*, pages 316-323, 1998.
9. R.Srikant, R.Agrawal. Mining Generalized Association Rules. In *Procs. of the 21st Int'l Conference on Very Large Databases*, 1995.
10. D.Tsur and others. Query Flocks: A Generalization of Association-Rule Mining. In *Procs. of ACM SIGMOD'98*, pages 1-12, 1998.
11. C.Zaniolo, H.Wang. Logic-Based User-Defined Aggregates for the Next Generation of Database Systems. In *The Logic Programming Paradigm: Current Trends and Future Directions*. Springer Verlag, 1998.
12. W. Shen, K. Ong, B. Mitbander, and C. Zaniolo. Metaqueries for Data Mining. In *Advances in Knowledge Discovery and Data Mining*, pages 375-398. AAAI Press/The MIT Press, 1996.