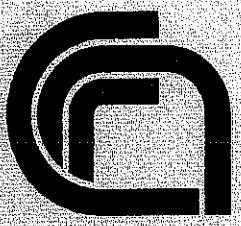

Consiglio Nazionale delle Ricerche

Integration of Deduction and Induction for Mining Supermarket Sales Data

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

Technical Report
CNUCE-B4-1999-01

CNUCE

Pisa

Integration of Deduction and Induction for Mining Supermarket Sales Data

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

January 15, 1999

Abstract

Datasift, a prototype system for the analysis of supermarket sales data based on data mining techniques, is presented. In market basket analysis, mining knowledge on customer behavior, which is actually useful to support marketing actions, is a difficult task, which requires non-trivial methods of employing and combining the data mining tools. To this purpose, we propose an architecture that integrates the deductive capabilities of a logic-based database language, LDL++, with the inductive capabilities of diverse data mining algorithms and tools, notably association rules. This paper presents an application of the integrated system to market basket analysis, and illustrates the high degree of expressiveness reached in dealing with high-level business rules.

1 MOTIVATION AND OBJECTIVES

Market basket analysis 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. Consider the chart of fig.1 (source: Databank), which shows the population of the light, medium and top classes of customers, compared to the economical relevance of the same classes. As an example, we can observe that less of 20% of customers are worth 80% of the company's revenue: a precise characterization of the high-spending classes of customers, as well as other classes and niches, is therefore an essential piece of knowledge to develop effective marketing strategies.

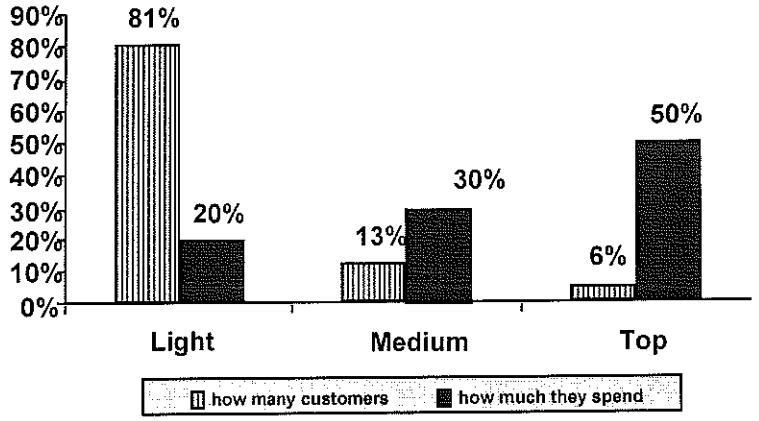

Figure 1: Distribution of Purchases.

However, 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* which represents the collection of items in a single purchase of a customer. A more effective approach is based on data mining algorithms and tools, notably *association rules*, which allow discovering relations between items purchased in the same basket [3, 2]. A famous example of association rule mined from supermarket sales data is that *beer* and *diapers* are often likely to be together in the same basket [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 association rules themselves.

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 market basket analysis. We briefly present Datasift, a prototype system for the analysis of supermarket sales data, based on an architecture 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 the various aspects of market basket analysis is discussed through a series of examples, covering data preparation, rule extraction, postprocessing, 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 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 DMQL [8] extends SQL with a collection of operators for mining different forms of rules. Other similar approaches include the M-SQL proposal [9], the *mine* operator of [11], as well as the *query flocks* proposal of [13]. However, we argue that combining 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. The examples discussed later in this paper show how this extra expressiveness is essentially due to the fact that deductive rules provide a uniform logic-based notation for data, queries, constraints, association rules and business rules, which support sophisticated forms of reasoning, and helps in bridging the gap between association rule mining and market basket analysis.

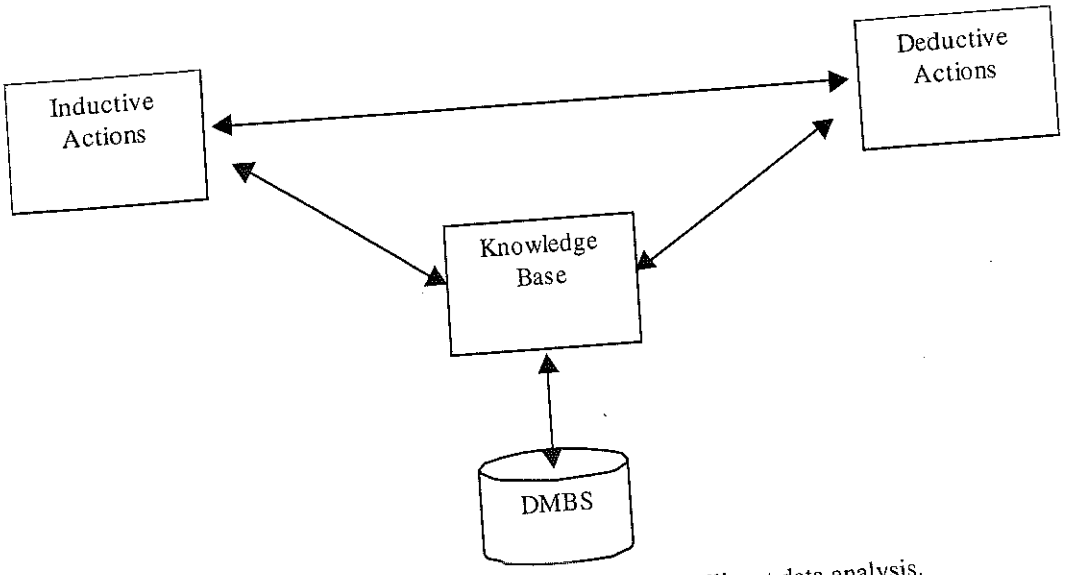

Figure 2: integration of deduction and induction for intelligent data analysis.

2 OVERVIEW OF SYSTEM ARCHITECTURE

The Datasift system adopts a hierarchical client-server web-based architecture, in which:

- the client component (*User Interface*) queries the knowledge base (by mean 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.)

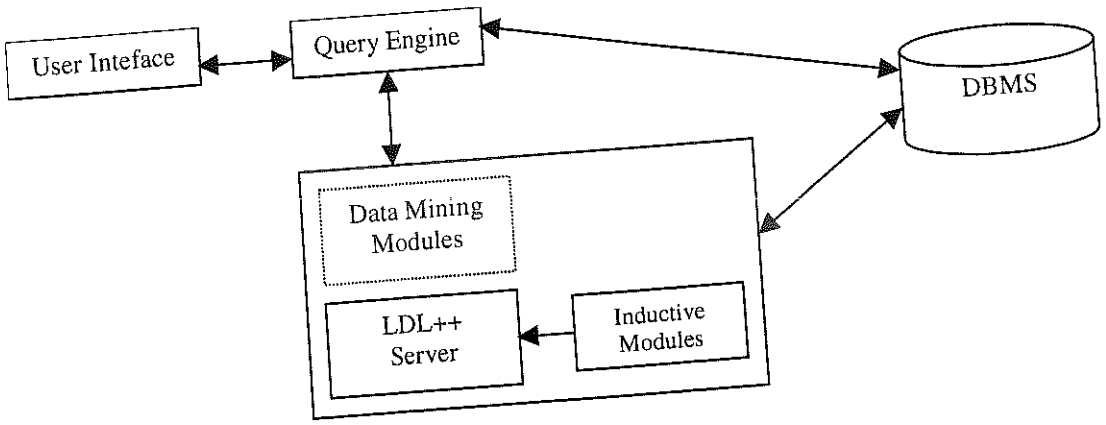

Figure 3: architecture of Datasift.

2.1 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 Microsoft 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.

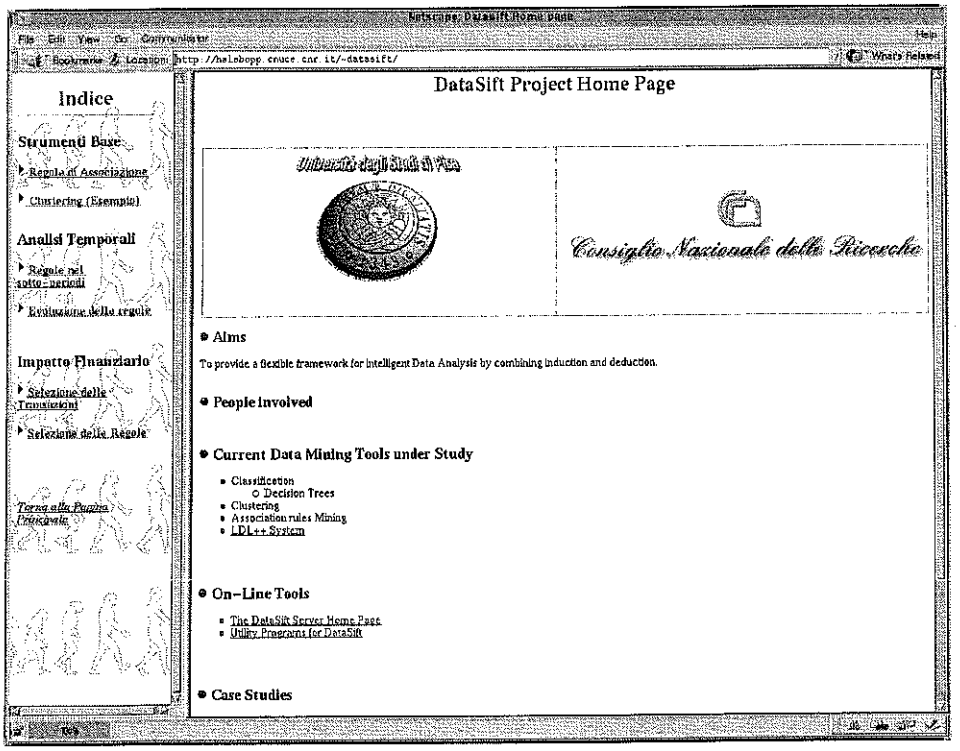

Figure 4: Web interface (main page).

An example snapshot of the current user interface is given in fig.4. On the left side a list of the available analyses is provided. This interface is trivially extendible e.g. by inserting java applets, which provides some navigation capabilities over output data.

An example of user interface for the input of parameters for the analysis is the following, used in the computation of association rules (whose meaning will be made clearer in later sections).

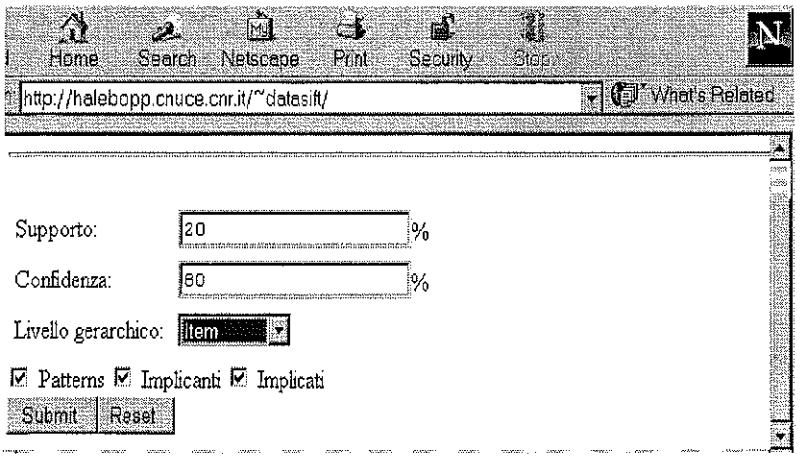

Figure 5: Parameters input interface (association rules page).

However, a more accurate representation metaphor is currently implemented by means of a Microsoft Excel application that, in addition to the web-based interaction capabilities previously described, allows us to use standard representation metaphors, whose flexibility is well known. 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.

2.2 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], that 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, based on a deductive database language.

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.

In Datasift we adopted the LDL++ deductive database system, which provides, in addition to the typical deductive features, a highly expressive query language [6, 15] obtained by introducing the non-deterministic *choice* operator and XY-stratified negation [14], still preserving its declarative nature [7].

A general way of dealing with data mining in a deductive framework is to directly define queries LDL++, that is particularly appropriate as a general purpose description language. For example, in order to define a typical (two-level) association rule query, the following LDL++ toy program can be defined:

```
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). For example, with 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)`.

From the expressiveness viewpoint, the LDL++ language is flexible enough to let define a degree of interestingness of the rules based not necessarily on the notions of support and confidence (as usually defined in the standard frameworks). Better, other interesting measures can be defined, such as financial measures or general rules that are not intended to model frequent marginals in a discrete (multinomial) probability distribution. 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.

In addition to the above-exemplified features, the LDL++ system has an open architecture, able to interface with a variety of external systems and components. This feature reveals very useful for integrating the core language with new functionalities, such as induction algorithms.
In particular, we propose a technique that allows using specialized algorithms (and hence specialized data structures), but still allows an easy integration with the language. The proposed architecture considers some inductive modules as aggregates, when the formal specification of the mining task is compatible with such hypothesis. For example, in the case of association rules, which we consider in this paper, we write rules of the form:

```
rules(patterns<(min_supp, min_conf,X1,…,Xn)>) ← q(X1,…,Xm).
```

where the aggregate `patterns` is implemented with some typical algorithm for the computation of the association rules, and `min_supp` and `min_conf` are required to be bound. The predicate `rules` is then instantiated by quadruples `(L,R,S,C)` where `L` and `R` represent the rule `L⇒R` with (normalized) support `S` and (normalized) confidence `C`, such that `S ≥ min_supp` and `C ≥ min_conf`.

The following program that computes the two-dimensional association rules gives a straightforward example:

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

Here, by querying `rules(X,Y,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)
```

In the last example, differently from the previous one, an ad hoc induction algorithm directly computes the patterns, and the deductive engine has the only task of exchanging the data to the inductive engine on demand.

The approach has two main advantages:

- the only modification on the underlying abstract machine is to provide an "open architecture", necessary to exchange information with the induction engine;
- the induction engine and the related optimizations are both transparent and independent from the deductive engine.

3 A MARKET BASKET ANALYSIS APPLICATION

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*.

3.1 Database schema

Figure 6 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.

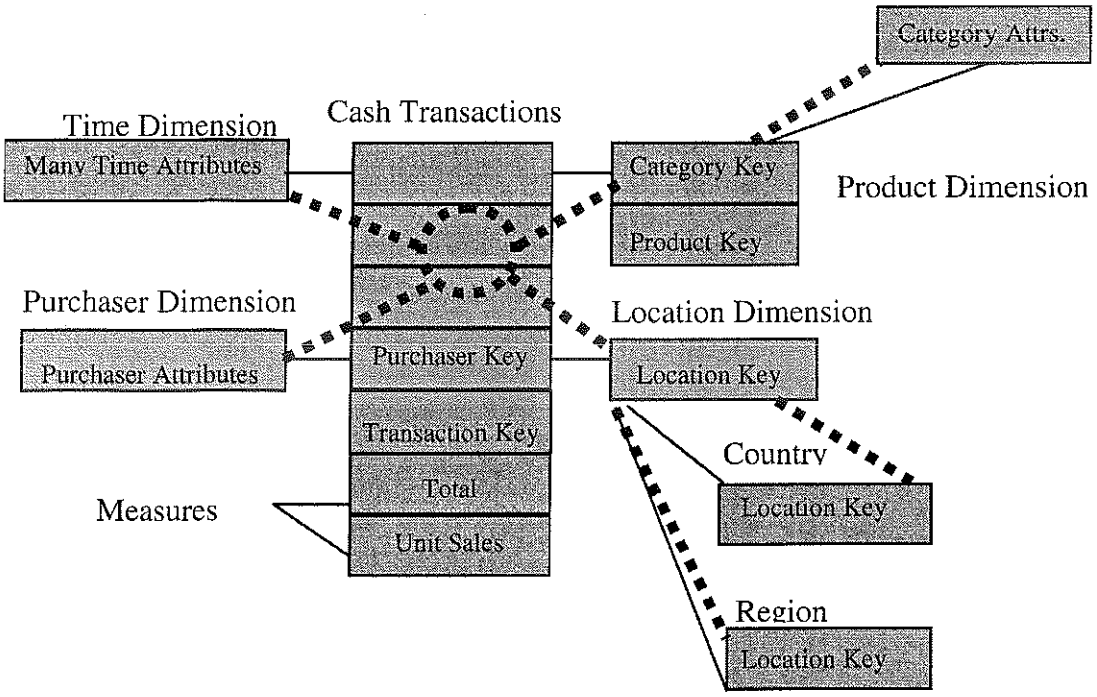

Figure 6: database schema.

3.2 Preprocessing: 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(label_1, 2/2/1998, 2/16/1998).
.
.
.
interval(label_n, 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 ≠ plastic bag`).

- *Hierarchies.*
  Usually, single items in a retail context are grouped into categories, such as *oranges*, *biscuits* and *milk*, which in turn are grouped in super-categories, such as *fruit*, *bakery* and *milk and derivatives*, and so on. In general, 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 [12].

  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).
```

3.3 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 in sect.2.2, 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. The picture in fig.7 displays the visualization performed by the client engine on a sample dataset.

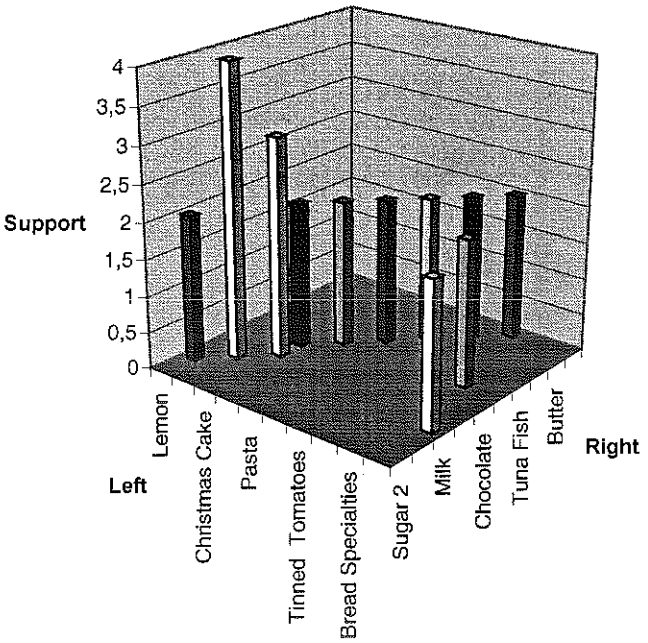

Figure 7: visualization of Association Rules

3.4 Postprocessing

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).
```

3.5 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.
```

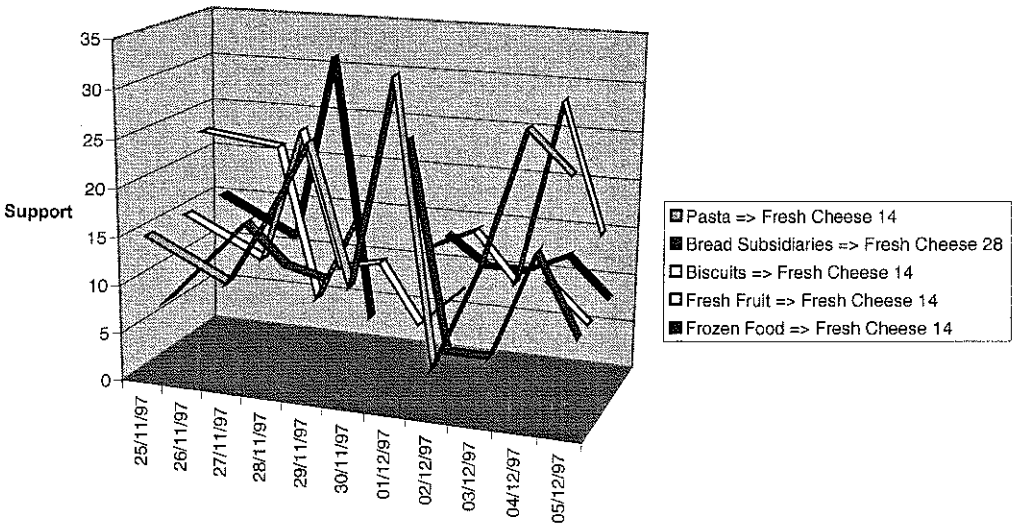

Figure 8: visualization of rules evolution.

- *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, …).
```

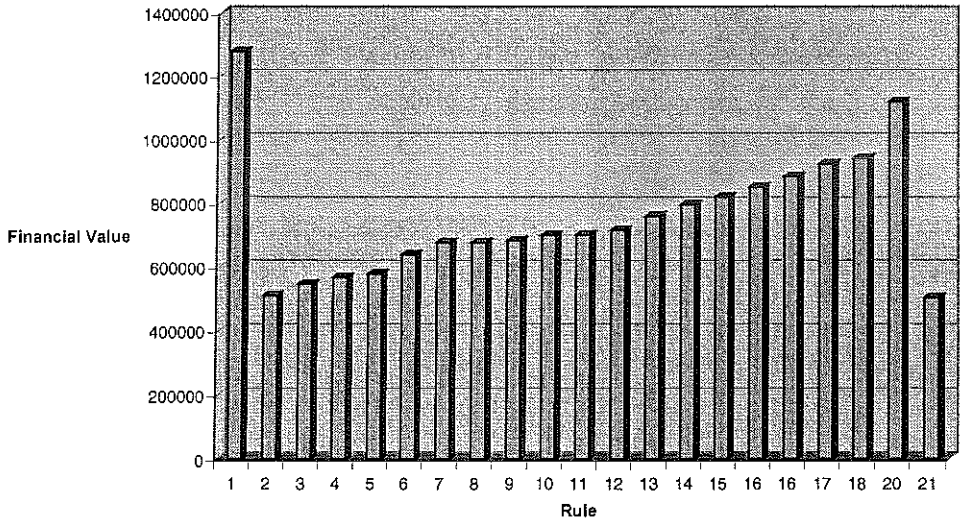

Figure 9: financial support of association rules.

4 CONCLUSIONS

In this paper we briefly presented a system for the analysis of supermarket sales data, 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 as an effective tool for the market analysts, which, using the predefined business rules, is 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.

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 modified in order to support such analysis.

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 future engineering of the Datasift system, we intend to adopt suitable data reduction techniques.

5 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.Arni, 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. F. Giannotti, G. Manco, M. Nanni, D. Pedreschi. On the Effective Semantics of Temporal, Non-monotonic, Non-deterministic Logic Languages. In Procs Int'l Conference on Computer Science Logic, LNCS, 1998.

8. 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.

9. T.Imielinski, A.Virmani, A.Abdulghani. Discovery Board Application Programming Interface and Query Language for Database Mining. In Procs.2nd Int'l Conference on Knowledge Discovery and Data Mining. 1996.

10. R.Kimball, K.Strehlo. Why Decision Support Fails and How to Fix it. SIGMOD Record, 24(3), 1995.

11. R.Meo, G.Psaila, S.Ceri. A Tightly-Coupled Architecture for Data Mining. In International Conference on Data Engineering (ICDE98), pages 316-323, 1998.

12. R.Srikant, R.Agrawal. Mining Generalized Association Rules. In Procs. of the 21st Int'l Conference on Very Large Databases, 1995.

13. D.Tsur and others. Query Flocks: A Generalization of Association-Rule Mining. In Procs. of ACM SIGMOD'98, pages 1-12, 1998.

14. C.Zaniolo, N.Arni, K.Ong. Negation and Aggregates in Recursive Rules: The LDL++ Approach. In Procs. 3rd Int. Conf. on Deductive and Object-Oriented Databases (DOOD93), volume 760 of LNCS, 1993.

15. 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.