

Data Mining

Elena Marchiori

Free University Amsterdam

Faculty of Sciences

Department of Mathematics and Computer Science

Amsterdam, The Netherlands

Abstract

In the recent past, many organizations have produced a large amount of machine-readable data in the form of files and databases. The explosive growth in data has generated an urgent need for techniques and tools for extracting useful hidden information from the data, in terms, e.g., of patterns or rules. Information from the data is used to analyze the user behaviour, to improve the services provided, and to increase the business opportunities. Data mining refers to the discovery process of hidden information from the data. This article presents a short overview of data mining goals, techniques, and applications.

Keywords: Data mining, knowledge discovery in databases, association rules, data generalization, machine learning, information management.

Short contents list 1 Introduction. 2. Goals 3. Techniques. 4. Applications. 5. Conclusion

Glossary of terms, abbreviations, symbols, etc KDD: Knowledge discovery in databases. DM: Data Mining. OLAP: On line analytical processing. ILP: Inductive logic programming. SQL: Query language for relational databases.

1 Introduction

In the 1980s, all major organizations built infra-structural databases, containing gigabytes of data about their clients, competitors and products. This

explosion of data in the modern society has produced a new source of hidden knowledge, consisting of regularities and patterns in the data.

In international organizations, this kind of knowledge provides an essential method for understanding the behaviour of the users in order to improve the services provided by the organization.

This justifies the sudden expansion and popularity of a new field called Knowledge Discovery in Databases (KDD). KDD is a multi-disciplinary field of research which exploits and integrates techniques from different areas, like machine learning, statistics, database technology, expert systems, and data visualization. These techniques are integrated for the non-trivial extraction of implicit, previously unknown and potentially useful knowledge from data in databases.

The main stages of the KDD process can be summarized as follows [1]:

1. *data cleaning*. Noisy, erroneous, missing, or irrelevant data are handled.
2. *data integration*. Multiple heterogeneous data sources may be integrated into one.
3. *data selection*. Data relevant to the analysis task are retrieved from the database.
4. *data coding*. Data are transformed into forms appropriate for extracting useful knowledge.
5. *data mining*. Advanced techniques are applied in order to extract useful knowledge.
6. *reporting*. Visualization and knowledge representation techniques are used to present the mined knowledge to the user.

A methodology for knowledge discovery in databases consists of the sequential application of the stages above enumerated; at every stage it is possible to jump back one or more stages. For instance, polluted data can be discovered during the application of the data coding or data mining stage, hence the process has to jump back to the data cleaning stage in order to eliminate the discovered pollution.

Observe that the term Data Mining (DM) refers to the discovery stage of the KDD process (stage 5 in the above description). However, many authors consider Data Mining and KDD as the same process.

The aim of this article is to introduce the reader to the field of data mining by describing its goals, techniques, and applications. For each of these topics a list of relevant issues is presented and each issue is briefly

described. The paper is mainly based on material from the following works [1, 3, 4, 5, 7, 8, 9, 11]. These works and those reported in the bibliography can be used by the reader who wishes to deepen his/her knowledge on knowledge discovery in databases.

2 Goals

The goals of data mining depend on the form of the data to be analyzed and on the type of information one is interested in. Following the treatment of this issue given in [3, 4, 6], one can distinguish the following main data mining tasks.

- *Class Description.* Data mining for class description finds a concise characterization of a collection of data and distinguishes it from others. For instance, class description can be used to compare European versus Asian sales of a company, identify the important factors which discriminate the two classes, and present a summarized overview.
- *Prediction.* The aim of predictive data mining is to find a description of how certain attributes within the data will behave in the future. For example, in business applications, the analysis of buying transactions to predict what consumers will buy under certain discounts, or how much sales volume a store would generate in a given period.
- *Identification.* Data patterns can be used to identify the existence of an event or an activity. For example, in biological applications, the existence of a gene may be identified by certain sequences of nucleotide symbols in the DNA sequence.
- *Association.* The discovery of association relationships or correlations among attributes of the data is a central task in data mining. For instance, data mining techniques may find rules of the form ‘98% of customers that purchase tires and automobile accessories also have automotive services carried out’. Association analysis is widely used in transaction data analysis for marketing, catalog design, and other business decision making processes.
- *Sequential pattern analysis.* A sequence of actions or events is sought. For example, if a patient underwent cardiac bypass surgery for blocked arteries and an aneurysm and later developed high blood urea within a year of surgery, then the patient is likely to suffer from kidney failure within the next 18 months. Detection of sequential patterns is

equivalent to detecting associations among events with certain temporal relationships.

- *Classification.* Data mining can partition the data so that different classes or categories can be identified based on combinations of attributes. For example, customers in a supermarket can be categorized into discount-seeking shoppers, shoppers in a rush, loyal regular shoppers, and infrequent shoppers.
- *Clustering.* A cluster is a collection of objects that are near each other with respect to a given similarity measure. Data mining for clustering identifies clusters embedded in the data. For instance, one may cluster the houses in an area according to their house category, floor area, and geographical locations.
- *Time-series analysis.* A large set of time-series data is analyzed to find certain regularities and interesting characteristics, including search for similar sequences, and mining sequential patterns, periodicities, trends and deviations. For example, a pattern in solar magnetic wind may be used to predict changes in earth atmospheric conditions.

3 Techniques

Data mining techniques are a ‘anything goes’ affair which uses approaches from multiple disciplines, like statistics, machine learning, information retrieval, and high performance computing. Different methods, like neural networks, evolutionary computation, pattern recognition, spatial data analysis, signal processing, probabilistic graph theory, and inductive logic programming, can be adapted and integrated into hybrid systems for data mining.

A large set of data analysis methods have been developed in statistics over many years of studies. Machine learning has also contributed substantially to classification and induction problems. Neural networks have shown their effectiveness in classification, prediction, and cluster analysis tasks. However, with increasing large amounts of data stored in databases for data mining, these methods face challenges on efficiency and scalability. Efficient data structures, indexing and data accessing techniques developed in database research contribute to high performance data mining.

This section reviews some of the most important machine-learning and pattern recognition algorithms that are used in data mining: query tools, visualization techniques, case-based learning (k-nearest neighbor), decision

trees, association rules, neural networks, genetic algorithms, and inductive logic programming.

3.1 Query Tools

The first step in mining a data set should always be a rough analysis of the data using traditional query tools.

For instance, by applying simple structured query languages, like SQL, one can obtain useful insight on basic aspects and structure of the data.

Observe that SQL is a structured language that assumes the user is aware of the database schema. It allows to view the same information along multiple dimensions, by means of operations of relational algebra that allow a user to select from tables (rows and columns of data) or to join related information from tables based on common fields.

As a consequence, with SQL one can uncover only shallow data, hence to retrieve information that is easily accessible from the data set. Thus SQL does not really belong to the data mining techniques. Nevertheless, most of the interesting information (around 80%) can be retrieved from the database using SQL. More sophisticated techniques are needed for mining the remaining interesting information (around 20%), which consists of hidden knowledge that can be of strategic importance for large organizations.

3.2 Visualization Techniques

Visualization techniques provide another tool that can be used at the beginning of the analysis of a data set in order to get a rough idea of the structure and distribution of the data.

For instance, an elementary technique that can be of great help for a preliminary data analysis is the so-called scatter diagram. In this technique, information on two attributes is displayed in a Cartesian space.

Scatter diagrams can be used to identify interesting sub-sets of the data set which can be further mined using more advanced techniques for extracting useful hidden information. The search for interesting projections of data sets constitutes a whole field of research, known as projection pursuit.

3.3 K-Nearest Neighbor

The representation of the records in the dataset as points in a multi-dimensional space is very useful for the analysis of the data. Using this representation, the concept of neighborhood can be defined, where records that are close to each other in the space are considered to belong to each other neighborhood. This

notion is used in a simple yet powerful learning technique, called k-nearest neighbor, where k denotes the number of neighbors that are used.

The basic idea of the k-nearest neighbor learning algorithm is ‘do as your neighbors do’. For instance, in order to predict the behaviour of a certain individual, the best k neighbors of that individual are considered, and the average of the behaviour of the neighbors provides the prediction for the behaviour of that individual.

The k-neighbor technique is an elegant and simple search method. However, it has a number of drawbacks which limit its general applicability.

For instance, the k-neighbor algorithm has a quadratic computational complexity (in the number of records of the data set) which prevents its application to very large datasets.

Another problem is related to the number of attributes of a record. A record consisting of many independent attributes is represented by a point in a high-dimensional search space. In high dimensional spaces, every two points have almost the same distance, thus the k-neighbor technique does not provide any useful information, since all pairs of points are neighbors.

Finally, the k-neighbor technique does not provide a theory to understand the structure of the data. This latter drawback can be overcome by the technique described in the next subsection.

3.4 Decision Trees

Decision trees represent Boolean functions. Given in input a record, a decision tree outputs a yes/no ‘decision’. Each internal node represents a test of the value of one of the attributes, and the branches from the node are labeled with the possible values of the test. Each leaf node in the tree specifies the Boolean value to be returned if that leaf is reached.

Decision tree induction is one of the simplest and yet most successful forms of learning algorithms.

Consider a classification problem for a target predicate. For instance, suppose we are given a database of a magazine publisher consisting of records containing attributes of the form age, income, credit, and binary attributes describing the purchase of the five types of magazines the publisher sells, namely cars, houses, sports, music, and comics.

Suppose the aim is to predict who will buy a cars magazine. Then the target predicate is ‘the customer will buy a cars magazine’. The target predicate is used to partition the database into two classes of examples: the class of positive examples, consisting of the records in which attribute cars magazine is true; and the class of negative examples, consisting of the records in which the attribute cars magazine is false.

The basic idea behind the decision tree induction is to test the most important attribute first, that is, the attribute that makes the most difference to the classification of the examples. For instance, suppose that the attribute sports magazine is present in 90% of the examples with value true (hence in the rest 10% of the examples with value false), while all other attributes are present only with 50% to 60% of the examples with value true. Then sports magazine is the most important attribute.

The most important attribute is used as the first test in the tree. Then for each value of that attribute one edge having that value as label is created, and the set consisting of records having that value for the chosen attribute is associated to that edge. In this way, the first attribute test splits up the dataset, and each outcome is a new decision learning problem in itself, with fewer records and one fewer attribute. One can distinguish three cases for these recursive problems.

1. The current data set contains either only positive examples or only negative examples (records having the same value for the cars magazine attribute). Then if all examples are positive a leaf node with decision yes is created, otherwise (if all examples are negative) a leaf node with decision no is created.
2. The current data set contains both positive and negative examples (records having different values for the cars magazine attribute).
 - (a) If there are attributes left then one can choose the most important attribute for that set in order to split the remaining records.
 - (b) Otherwise, it means that there is *noise* in the data, since the records in that set have the same description but different classifications.
3. The current data set is empty, meaning that there is no evidence for that attribute value. In such a case, a default value can be calculated from the majority classification at the node's parent and returned as decision.

There are many efficient algorithms for decision tree induction having $O(n \log(n))$ computational complexity, where n is the number of records in the initial dataset. Decision tree induction algorithms scale up very well for large data sets. Another advantage is that they provide a neat description of the nature of the decision process in logic. However, in some cases the induced decision trees may have exponential size in the number of attributes. For instance, this happens for decision trees approximating the parity function,

which returns 1 if and only if an even number of inputs are 1. In general, any kind of representation used is good for some kinds of functions, and bad for others.

3.5 Association Rules

Association rules are statements of the form: 98% of customers that purchase sports magazine also purchase cars magazine. These kinds of description provide clear customer profiles that can be used for marketing decisions.

Formally, an association rule is written as $X \Rightarrow Y \mid (c, s)$. Here X and Y are sets of binary attributes called *itemsets*, s.t. $X \cap Y = \emptyset$; c is the confidence of the rule; and s is the support of the rule. The *confidence* is a measure of the rule strength, that is, the percentage of records with all attributes in Y having value true within records with all attributes in X with value true. The *support* is a measure of the statistical significance of the rule, that is, the percentage of records that have all attributes in $X \cup Y$ with value true.

For instance, consider a market basket data set whose records describe the list of the products bought in a supermarket, like ‘milk, butter, bread, soap’. Then the association rule $\{Bread, Milk\} \Rightarrow \{Juice\} \mid (98, 70)$ says that 70% of all the records contain Bread, Milk, and Juice, and 98% of the records containing Juice contain also Bread and Milk.

Mining association rules in databases has attracted a lot of attention in the KDD research community. The goal is to generate all possible rules that exceed some minimum user-specified support s and confidence c threshold. The problem is decomposed into two steps:

1. Generate all itemsets that have support greater than the threshold s . Such itemsets are called large itemsets.
2. For each large itemset, generate all the rules that have confidence greater than the threshold c .

The second problem can be solved as follows: for a large itemset X and for a subset Y of X ($Y \subset X$), consider the set $X' = X \setminus Y$ consisting of those elements of X that are not in Y . Then generate the rule $X' \Rightarrow Y$ if the support of X divided by the support of X' is greater than c . The support of a itemset X is the number of records in the data set with all attributes in X having value true.

Generating association rules by using all large itemsets is relatively simple. However, discovering all large itemsets together with the value for their

support is a major problem if the cardinality of the set of items is very high. A typical supermarket has thousands of items. The number of distinct itemsets is 2^m , where m is the number of items, hence counting the support for all possible itemsets is computationally intensive.

To reduce the combinatorial search space, algorithms for finding association rules exploit the following properties of large itemsets:

- A subset of a large itemset is also large.
- Conversely, an extension of a non-large itemset is also non-large.

These properties are used in the basic algorithms for finding all large itemsets, like Apriori [2], whose main scheme can be summarized as follows [4].

1. Test the support for itemsets of size 1, called 1-itemsets, by scanning the database. Discard 1-itemsets having support smaller than s .
2. Extend the large 1-itemsets into 2-itemsets by adding one item of a large 1-itemset each time, to generate all candidate itemsets with two elements. Test the support of the generated candidates and discard all those 2-itemsets having support smaller than s .
3. Repeat the above steps; at step k , the previously found $(k-1)$ large itemsets are extended into k -itemsets and tested for minimum support.

The process is repeated until no new large itemsets can be found. Several algorithms based on this scheme have been introduced, which vary mainly in the way candidate itemsets are generated and in the way the supports for the candidate itemsets are counted. Moreover, various other kinds of association rules have been introduced: generalized association rules, which take into account the presence of taxonomies (*is-a* hierarchies) over the items; profile association rules, which describe associations between customer profile and behaviour information; constraint-based association rules, which are rules offering a predictive advantage over any of their simplifications; and negative association rules, which are rules of the form ‘60% of the customers who buy potato chips do not buy fruit’.

3.6 Inductive Logic Programming

A drawback of association rules is that they have the expressive power of propositional logic. Thus they cannot describe complex structured objects

and relations among objects or their components. This limitation of association rules is overcome by inductive logic programming.

Inductive logic programming (ILP) can be viewed as machine learning in a first-order language. It is relevant for knowledge discovery in databases because it can describe patterns involving more than one relation and it can incorporate domain-specific background knowledge. For instance, if the problem is to learn about properties of chemical compounds, the molecular structures can be introduced as background knowledge in terms of the atoms and bonds between them. The price for these benefits of ILP includes ILP's greater logical and computational complexity [9].

ILP is a highly technical field relying on advanced material from the study of computational logic. In the following two subsections we illustrate the two main approaches used in ILP systems by means of examples, mainly based on [11].

3.6.1 Inverse Resolution

The first approach uses techniques based on inverting a resolution proof, where resolution is a complete inference procedure for first-order logic. The following example from [11] illustrates this approach.

Suppose a description about family relationships is given, which includes facts like

Parent(Elizabeth, Anne),
Father(Philip, Anne),
Mother(Mum, Margaret),
Married(Elizabeth, Philip),
Male(Philip),
Female(Beatrice).

Suppose the goal is to learn the target concept Grandparent, for which a set of positive and negative examples is given:

Grandparent(Mum, Charles),
Grandparent(Elizabeth, Beatrice),
Grandparent(George, Anne),
... ,
 \neg Grandparent(Mum, Harry),
 \neg Grandparent(Spencer, Peter),
... .

The first steps of a backward proof are given below:

1. Start from the contradictory clause C1: True \Rightarrow False.

2. Use the positive example $\text{Grandparent}(\text{George}, \text{Anne})$ as goal, that is, add the clause C2: $\text{Grandparent}(\text{George}, \text{Anne}) \Rightarrow \text{False}$.
3. Apply inverse resolution to C1 and C2 to infer the clause C3: $\text{True} \Rightarrow \text{Grandparent}(\text{George}, \text{Anne})$.
4. Use the background information $\text{Parent}(\text{Elizabeth}, \text{Anne})$ as clause C4: $\text{True} \Rightarrow \text{Parent}(\text{Elizabeth}, \text{Anne})$. Apply inverse resolution to C3 and C4 to infer the clause C5: $\text{Parent}(\text{Elizabeth}, y) \Rightarrow \text{Grandparent}(\text{George}, y)$, where y is a fresh variable.

Step 4 is an example of generalization, where the inverse substitution which binds Anne to the variable y is used. Inverse resolution involves search, since each inverse resolution step is nondeterministic. For instance, in step 4 of the above backward proof fragment, the clause $\text{Parent}(\text{Elizabeth}, \text{Anne}) \Rightarrow \text{Grandparent}(\text{George}, \text{Anne})$ can also be inferred from C3 and C4 using inverse resolution, where no generalization is performed.

An exhaustive search process for inverse resolution would be extremely inefficient. Therefore ILP systems use a number of restrictions to make the process more manageable. An example of such a restriction states that the arguments of the predicates may not contain function symbols.

3.6.2 Top Down Learning

The second approach to ILP is essentially a generalization of the techniques for decision trees learning to the first-order logic case. Rather than starting from the observations and working backwards, one can start with a very general rule that is gradually specialized in order to fit the data. This method is similar to the way decision trees are incrementally built. In the first-order case, first-order literals are used instead of attributes, and the learning process extracts a set of clauses instead of a decision tree.

In order to illustrate this approach, consider again the family relationships description together with the classification examples for the target predicate Grandparent given in the previous subsection.

In order to learn the predicate Grandparent , one can proceed as follows:

1. Start with the clause C1: $\Rightarrow \text{Grandparent}(x, y)$, with x and y variables. Note that this clause classifies all examples as positive, thus it has to be specialized in order to rule out negative examples.
2. A specialization of C1 is the clause C2: $\text{Father}(x, z) \Rightarrow \text{Grandparent}(x, y)$, with z fresh variable.

3. Clause C2 has to be further specialized, in order to rule out the cases in which x is the father of z but z is not a parent of y , cases which are described by the negative examples. Adding the single literal $\text{Parent}(z,y)$ to C2 gives the clause C3: $\text{Father}(x,z) \wedge \text{Parent}(z,y) \Rightarrow \text{Grandparent}(x,y)$, which correctly classifies all the examples.

The preceding example is a very simple illustration of FOIL [10], one of the first programs using this approach.

3.7 Neural Networks

ILP is a symbolic approach to learning, because it employs a representation based on first-order logic. Neural networks are based on a completely different approach called quantitative. Effective systems based on both approaches have been developed, and the choice of which approach to choose depends on the task to be solved, as well as on the user background.

The material of this section is based on [11].

Neural networks are machines which model the way in which the human brain performs a particular task or function of interest. A neural network consists of a number of nodes, or *units*, connected by *links*. Each link has a numeric weight associated with it.

Weights are the primary means of long-term storage in neural networks, and learning usually takes place by updating the weights. Some of the units are connected to the external environment, and can be designed as input or output units.

The weights are modified so as to try to bring the network's input/output behaviour more in accordance with the behaviour of the environment providing the inputs. Each unit has a set of input links from other units, a set of output links to other units, an activation level and a function for computing the activation level at the next step in time, given the inputs and weights.

The idea is that each unit does a computation based on inputs from its neighbors, but without need for any global control over the set of units as a whole. In practice, most neural networks implementations are in software and use synchronous control to update all the units in a fixed sequence. To build a neural network to perform some task, one must first decide how many units are to be used, what kind of units are appropriate, and how the units are to be connected to form a network. Moreover, the use of examples also implies that one must decide how to encode the examples in terms of inputs and outputs of the network.

Neural networks self-adapt, that is, they learn from information on a specific problem. They perform well on classification tasks and are therefore

useful in data mining. However, they do not provide a neat representation of the model they learn, hence it is in general difficult to understand what a neural network has learnt. Despite this major drawback, neural networks are popular and frequently used in commercial data mining systems.

3.8 Genetic Algorithms

Genetic algorithms are modeled after the adaptive emergence of biological species from evolutionary mechanisms. They are a class of population based stochastic procedures capable of adaptive and robust search over a wide range of problems search spaces. The construction of a genetic algorithm involves devising an alphabet that encodes the candidate solutions to the problem in terms of strings of that alphabet. Strings represent individuals. A fitness function specifies which individuals can survive and which cannot. Individuals are combined using the crossover operator, which merges two strings (called parents) in order to produce two new strings (called offsprings). Moreover, offsprings may undergo random mutations, where a small portion of the string is modified. The crossover and mutation operators are applied to chromosomes with a user-given probability, called crossover rate and mutation rate, respectively.

In order to set up a genetic algorithm for a given problem one has to specify the following main features: the representation of candidate solutions; the fitness function; the selection mechanism; the reproduction operators.

In a standard GA, individuals are represented as strings over a finite alphabet, and each element of the string is called gene. Nevertheless, GAs using alternative representations are becoming more and more common, and are called genetic programs when the representation has a tree like structure. The fitness function depends on the problem: it takes as input a chromosome and gives as output a real number describing how good the chromosome is. The selection strategy is usually randomized: in the standard selection strategy the probability of selection is proportional to the fitness. Reproduction is accomplished by cross-over and mutation. For instance, in one-point crossover, a crossover point is randomly chosen for a pair of individuals selected for reproduction, which divides each of the two chromosomes in two parts. Then two offsprings are created by taking the left (respectively right) part from one parent and the right (respectively left) part from the other one. Moreover, with small probability each of the offsprings can be mutated, by changing the value of one of its genes.

4 Applications

Data mining technologies can be applied to a large variety of decision-making contexts in business. In particular, areas of significant payoffs are expected to include the following [4, 6]:

- **Marketing.** Applications include analysis of consumer behaviour based on buying patterns; determination of marketing strategies including advertising, store location, and targeted mailing; segmentation of customers, stores, or products; and design of catalogs, store layouts, and advertising campaigns.
- **Finance.** Applications include analysis of credit-worthiness of clients, segmentation of account receivables, performance, analysis of finance investments like stocks, bonds, and mutual funds; evaluation of financing options; and fraud detection.
- **Manufacturing.** Applications involve optimization of resources like machines, manpower, and materials; optimal design of manufacturing processes, shop-floor layouts, and product design, such as automobiles based on customer requirements.
- **Health Care.** Applications include an analysis of effectiveness of certain treatments; optimization of processes within hospital, relating patient wellness data with doctor qualifications; and analyzing side effects of drugs.

5 Conclusion

The diversity of data, data mining tasks, and data mining techniques poses many challenging issues [6]. Important tasks for data mining researchers and data mining system developers include: the design of data mining languages, the development of efficient and effective data mining algorithms and systems, the construction of integrated data mining environments, and the application of data mining techniques for solving large real life problems.

With the fast computerization of the society, the social impact of data mining is of fundamental importance. When a large amount of interrelated data are effectively analyzed from different perspectives, it can pose threats to the goal of protecting data security and guarding against the invasion of privacy. It is a challenging task to develop effective techniques for preventing the disclosure of sensitive information in data mining, especially as the use of

data mining systems is rapidly increasing in domains ranging from business analysis, customer analysis, to medicine and government.

This survey on data mining techniques is intended as a brief introduction to the topic. The reader interested in the subject is referred to the books and articles listed in the References. In particular, the recent book [12] covers all the major topics in data mining with algorithms and implementations in Java. Moreover, the WWW address www.kdnuggets.com contains interesting on-line material as well as pointers to academic and commercial systems for data mining.

References

- [1] P. Adriaans and D. Zantinge. *Data Mining*. Addison-Wesley, 1996.
- [2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases*, pages 478–499, 1993.
- [3] M.S. Chen, J. Han, and P.S. Yu. Data mining: An overview from a database perspective. *IEEE Transactions on Knowledge and Data Engineering*, 8(6):866–883, 1996.
- [4] R. Elmasri and S.B. Navathe. *Fundamentals of Database Systems*. Addison-Wesley, 2000.
- [5] U. Fayyad, G. Piatesky-Shapiro, P. Smyth, and R. Uthurusamy (eds.). *Advances in Knowledge Discovery and Data Mining*. AAAI/MIT Press, 1996.
- [6] J. Han. Data mining. *Encyclopedia of Distributed Computing*, 1999.
- [7] M. Holsheimer and A.P.J.M. Siebes. Data mining: the search for knowledge in databases. Technical report, CWI, CS-R9406 1994.
- [8] N. Lavrač and S. Džeroski. *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, 1994.
- [9] R.S. Michalski, I. Bratko, and M. Kubat (eds.). *Machine Learning and Data Mining: Methods and Applications*. Wiley, England, 1998.
- [10] J. R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5(3):239–266, 1990.

- [11] S. Russell and P. Norvig. *Artificial Intelligence: a Modern Approach*. Prentice-Hall International, 1995.
- [12] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, 1999.