

Towards a General Framework for Data Mining

Sašo Džeroski

Jožef Stefan Institute, Jamova 39, 1000 Ljubljana, Slovenia

Abstract. In this paper, we address the ambitious task of formulating a general framework for data mining. We discuss the requirements that such a framework should fulfill: It should elegantly handle different types of data, different data mining tasks, and different types of patterns/models. We also discuss data mining languages and what they should support: this includes the design and implementation of data mining algorithms, as well as their composition into nontrivial multi-step knowledge discovery scenarios relevant for practical application. We proceed by laying out some basic concepts, starting with (structured) data and generalizations (e.g., patterns and models) and continuing with data mining tasks and basic components of data mining algorithms (i.e., refinement operators, distances, features and kernels). We next discuss how to use these concepts to formulate constraint-based data mining tasks and design generic data mining algorithms. We finally discuss how these components would fit in the overall framework and in particular into a language for data mining and knowledge discovery.

1 Introduction: The Challenges for Data Mining

While knowledge discovery in databases (KDD) and data mining have enjoyed great popularity and success in recent years, there is a distinct lack of a generally accepted framework for data mining. The present lack of such a framework is perceived as an obstacle to the further development of the field. For example, at the SIGKDD-2003 conference panel “Data Mining: The Next 10 Years” (Fayyad et al. 2003), U. Fayyad emphasizes in his position statement that “the biggest stumbling block from the scientific perspective is the lack of a fundamental theory or a clear and well-understood statement of problems and challenges”.

Yang and Wu (2006) collected the opinions of a number of outstanding data mining researchers about the most challenging problems in data mining research (and presented them at ICDM-2005). Among the ten topics considered most important and worthy of further research, the development of a unifying theory is listed first. The article states: “Several respondents feel that the current state of the art of data mining research is too ad-hoc. ... a theoretical framework that unifies different data mining tasks ..., as well as different data mining approaches ..., would help the field and provide a basis for future research.”

High on the list of important research topics is mining complex data (Yang and Wu 2006). We will take complex data here to mean structured data that depart from the format most commonly used in data mining, namely the format

of a single table. This will include sequences and graphs, but also text, images, video, and multi-media data. From this viewpoint, much of the current research in data mining is about mining complex data, e.g., text mining, link mining, mining social network data, web mining, multi-media data mining. Many of the grand challenges for data mining are perceived to be in this area, cf. the SIGKDD-2006 conference panel (Piatetsky-Shapiro et al. 2006). An additional challenge is to treat the mining of different types of structured data in a uniform fashion, which is becoming increasingly more difficult, as somewhat separate research (sub)communities are evolving around text/link/tree/graph mining.

Hand in hand with the above go the problems of mining complex patterns and the incorporation of domain knowledge (Yang and Wu 2006; Aggrawal's comments in Fayyad et al. 2003). As the complexity of the data analyzed grows, more expressive formalisms are needed to represent patterns found in the data. The use of such formalisms has been proposed within relational data mining (Džeroski and Lavrač 2001) and statistical relational learning (Getoor and Taskar, 2007); these are now used increasingly more often in link mining, web mining and mining of network data.

The last set of methodological questions that we want to emphasize here concerns knowledge discovery as a process, rather than individual data mining tasks and approaches. Data preparation typically takes significant time and different data mining operations need to be applied and composed in practical applications. Arguably, there is insufficient support for humans carrying out the knowledge discovery process as a whole. Integration and compositionality of data mining operations/algorithms are called for.

At a KDD-03 panel discussion (Fayyad et al. 2003), Fayyad states: "In a typical data mining session, I spend most of my time extracting and manipulating data, not really doing data mining and exploration. The trail of 'droppings' I leave behind in any given data mining session is enormous, and it seems every time it is replicated and repeated, almost from scratch, again." Ramakrishnan (at the same panel) follows suit and identifies as technical challenges the following: "(1) Finding ways to address the real bottleneck in data mining, which is the human cycles spent in analyzing data. ... Real advances will come from techniques that lead to more efficient management of the process of data mining, and that reduce the cycle time in arriving at useful insights. (2) Data mining is often perceived as a bag of tricks. We need to at least provide a vision of how these tricks fit into a coherent tool-kit." In the same article, Uthurusamy makes the point that: "Even obvious and immediate needs like ... the core need of integration have not received their much-deserved attention... The original process centric view of KDD espoused the three 'I's (Integrated, Iterative, and Interactive) as basic for KDD. These are central to the ideas of 'Computer Assisted Human Discovery' and 'Human Assisted Computer Discovery.' There has been very little work on these in recent years." Yang and Wu (2006) also point out the need to support the composition of data mining operators, as well as the need to have a theory behind this.

In the remainder of this article, we first discuss (Section 2) inductive databases and inductive queries, one of the most promising approaches to formulating a general framework for data mining, then discuss the desirable properties of such a framework (Section 3). Section 4 defines the basic concepts of data mining, including data, patterns/models, and data mining tasks. Section 5 discusses the dual nature of patterns/models, which can be viewed both as data and as functions. Section 6 introduces constraint-based data mining and discusses the types of constraints considered therein. Section 7 introduces the key ingredients of data mining algorithms (i.e., refinement operators, distances, features and kernels). Section 8 revisits constraint-based data mining and treats it in the context of the basic ingredients from Section 7. Section 9 discusses the design of generic data mining algorithms for structured data. We finally discuss how all of the above components would fit in the overall framework and in particular into a language for data mining and knowledge discovery in Section 10. The article closes with a brief discussion of related work.

2 Inductive Databases and Inductive Queries

Inductive databases (IDBs, Imielinski and Mannila 1996, De Raedt 2002a) are an emerging research area at the intersection of data mining and databases. In addition to normal data, inductive databases contain patterns (either materialized or defined as views). Besides patterns (which are of local nature), models (which are of global nature) can also be considered. In the IDB framework, patterns become “first-class citizens” and can be stored and manipulated just like data in ordinary databases.

Inductive databases embody a database perspective on knowledge discovery, where knowledge discovery processes become query sessions. Ordinary queries can be used to access and manipulate data, while inductive queries (IQs) can be used to generate (mine), manipulate, and apply patterns. KDD thus becomes an extended querying process (Imielinski and Mannila 1996) in which both the data and the patterns that hold (are valid) in the data are queried. IDB research thus aims at replacing the traditional KDD process model, where steps like pre-processing, data cleaning, and model construction follow each other in succession, by a simpler model in which all data pre-processing operations, data mining operations, as well as post-processing operations are queries to an inductive database and can be interleaved in many different ways.

Given an inductive database that contains data and patterns, several different types of queries can be posed. Data retrieval queries use only the data and their results are also data: no pattern is involved in the query. In IDBs, we can also have cross-over queries that combine patterns and data in order to obtain new data, e.g., apply a predictive model to a dataset to obtain predictions for a target property. In processing patterns, the patterns are queried without access to the data: this is what is usually done in the post-processing stages of data mining. Data mining queries use the data and their results are patterns: new patterns are generated

from the data and this corresponds to the traditional data mining step. When we talk about inductive queries, we most often mean data mining queries.

A general statement of the problem of data mining (Mannila and Toivonen 1997) involves the specification of a language of patterns and a set of constraints that a pattern has to satisfy. The latter can be divided in two parts: language constraints and evaluation constraints. The first part only concerns the pattern itself, while the second part concerns the validity of the pattern with respect to a given database. Constraints thus play a central role in data mining and constraint-based data mining is now a recognized research topic (Bayardo 2002). The use of constraints enables more efficient induction and focusses the search for patterns on patterns likely to be of interest to the end user.

In the context of inductive databases, inductive queries consist of constraints. Inductive queries can involve language constraints (e.g., find association rules with item A in the head) and evaluation constraints, formed by using evaluation functions. The latter express the validity of a pattern on a given dataset. We can use these to form evaluation constraints (e.g., find all item sets with support above a threshold) or optimization constraints (e.g., find the 10 association rules with highest confidence).

Different types of data and patterns have been considered in data mining, including frequent itemsets, episodes, Datalog queries, and graphs. Designing inductive databases for these types of patterns involves the design of inductive query languages and solvers for the queries in these languages, i.e., constraint-based data mining algorithms. Of central importance is the issue of defining the primitive constraints that can be applied for the chosen data and pattern types, that can be used to compose inductive queries. For each pattern domain (type of data, type of pattern, and primitive constraints), a specific solver is designed, following the philosophy of constraint logic programming (De Raedt 2002b).

The IDB framework is an appealing approach towards a theory for data mining, because it employs declarative queries instead of ad-hoc procedural constructs. As such, it holds the promise of facilitating the formulation of an “algebra” for data mining, along the lines of Codd’s relational algebra for databases (Calders et al. 2006b, Johnson et al. 2000). The IDB framework is also appealing for data mining applications, as it supports the entire KDD process (Boulicaut et al. 1999). In inductive query languages, the results of one (inductive) query can be used as input for another: nontrivial multi-step KDD scenarios can be thus supported in IDBs, rather than just single data mining operations.

The state-of-the-art in IDBs (Boulicaut et al. 2006) is that there exists some theory for and various effective approaches to constraint-based mining (inductive querying) of local patterns, such as frequent itemsets and sequences. There is an obvious lack of a theory for and practical approaches to inductive querying of global models. This issue has only recently began to attract some attention through the research on constrained induction of tree-based (Garofalakis et al. 2003, Struyf and Džeroski 2006) and equation-based (Džeroski et al. 2006) predictive models.

More importantly, most of the existing approaches to constraint-based data mining and inductive querying work in isolation and are not integrated with databases or other data mining tools. Only few attempts at integration have been made, such as the approach of mining views (Calders et al. 2006a). Answering complex inductive queries that involve different pattern domains and supporting complex KDD scenarios has also barely been studied.

3 Desiderata for a General Data Mining Framework

In this section, we briefly discuss the requirements that a general framework for data mining should fulfill. In our opinion, such a framework should elegantly handle different types of data, different data mining tasks, and different types of patterns/models. These should be orthogonal dimensions, so that combinations should be facilitated, e.g., tree-based approaches for classification of images (where the type of data is image(s), the task is classification and the model type is decision tree(s)).

One of the distinguishing features of data mining is its concern with analyzing different types of data. Besides data in the format of a single table, which is most commonly used in data mining, complex data are receiving increasing amounts of interest. These include data in the form of sequences and graphs, but also text, images, video, and multi-media data. Much of the current research in data mining is about mining such complex data, e.g., text mining, link mining, mining social network data, web mining, multi-media data mining. A major challenge is to treat the mining of different types of structured data in a uniform fashion.

Many different data analysis tasks have been considered so far within the field of data mining. By far the most common is the task of predictive modelling, which includes classification and regression. Mining frequent patterns is the next most popular, with the focus shifting from mining frequent itemsets to mining frequent patterns in complex data. Clustering, which has strong roots in the statistical community, is also commonly encountered in data mining, with distance-based and density-based clustering as the two prevailing forms. A variety of other tasks has been considered, such as change and deviation detection, but it is not clear whether these are of fundamental nature or can be defined by composing some of the tasks listed here.

Finally, different kinds/representations of patterns/models may be used for the same data mining task. This is most obvious for predictive modelling, where a variety of methods/approaches exist, ranging from rules and trees, through support vector machines, to probabilistic models (such as Naive Bayes or Bayesian networks for classification). The different types of models are interpreted in different ways, and different algorithms may exist for building the same kind of model (cf. the plethora of algorithms for building decision trees).

A general framework for data mining should define a set of basic concepts that cover the dimensions outlined above. Through combining these basic concepts, one should be able to obtain most of the diversity present in data mining approaches today. Hopefully, it would also facilitate the derivation of new approaches and

insights. The basic concepts would be a keystone in the development of data mining languages, which should support the design and implementation of data mining algorithms, as well as their composition into nontrivial multi-step knowledge discovery scenarios relevant for practical application. In the latter case, we can speak of knowledge discovery languages.

4 The Basic Concepts of Data Mining

“Knowledge discovery in databases (KDD) is the non-trivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data”, state Fayyad et al. (1996). According to this definition, data mining (DM) is the central step in the KDD process concerned with applying computational techniques (i.e., data mining algorithms implemented as computer programs) to actually find patterns in the data. To arrive at a general theory/framework for data mining, we need to have general definitions for the above terms, including data, patterns and validity.

The basic concepts of data mining include data, data mining tasks, and patterns/models. The validity of a pattern/model on a given set of data is related to the data mining task considered. Below we discuss these in some detail.

4.1 Data

Let us start with data: This is the most basic ingredient of data mining. A data mining algorithm takes as input a set of data. An individual datum in the data set has its own structure, e.g., consists of values for several attributes, which may be of different types or take values from different ranges. We typically assume that all data items are of the same type and share the same structure.

More generally, we are given a data type T and a set of data D of this type. We will not discuss in detail what a data type is and how to formally define it: any standard textbook on data structures (e.g., Aho et al. 1983) covers this topic, while a more formal treatment in the context of logic is given by Lloyd (2003). It is important to notice, though, that a set of basic/primitive types is typically taken as a starting point, and more complex data types are built by using type constructors. As argued above, it is of crucial importance to be able to deal with structured data, as these are attracting an increasing amount of attention within data mining.

Assume we are given a set of primitive data types, such as Boolean or Real. Other primitive data types might include Discrete(S), where S is a finite set of identifiers, or Integer. In addition, we are given some type constructors, such as Tuple and Set, that can be used to construct more complex data types from existing ones. For example, Tuple(Boolean,Real) denotes a data type where each datum consists of a pair of a Boolean value and a real number, while Set(Tuple(Boolean,Real)) denotes a data type where each datum is a set of such pairs.

Other type constructors might include Sequence(T), which denotes a sequence of objects of type T , or LabeledGraph(VL,EL), which denotes a graph where

vertex labels are of type VL and edge labels are of type EL. With these, we can easily represent the complex data types that are of practical interest. For example, DNA sequences would be of type $\text{Sequence}(\text{Discrete}(\{A, C, G, T\}))$, while molecules would be labeled graphs with vertices representing atoms and edges representing bonds between atoms: atoms would be labeled with the type of element (e.g., nitrogen, oxygen) and edges would be labeled with the type of bond (e.g., single, double, triple).

4.2 Patterns and Models

Here we will consider four types of patterns/models, which are directly related to the data mining tasks discussed later in this section. These are probability distributions, patterns (in the sense of frequent patterns), predictive models and clusterings. All of these are defined on a given type of data, except for predictive models, which are defined on a pair of data types. Note that we allow arbitrary (arbitrarily complex) data types. The most typical case in data mining would consider a data type $T = \text{Tuple}(T_1, \dots, T_k)$, where each of T_1, \dots, T_k is Boolean, Discrete(S) or Real.

A **probability distribution D on type T** is a mapping from objects of type T to non-negative Reals, i.e., has the signature $\mathbf{d} :: T \rightarrow \mathbf{R}^{0+}$. For uncountably infinite types, probability densities are used instead. The sum of all probabilities (the integral of the probability densities) over T is constrained to amount to one.

A **pattern P on type T** is a Boolean function on objects of type T, i.e., has the signature $\mathbf{p} :: T \rightarrow \mathbf{bool}$. A pattern on type T is true or false on an object of type T. Frawley et al. (1991) define a pattern as a statement (expression) in a given language, that describes (relationships among) the facts in (a subset of) the data. In the broadest sense, the word pattern is used to describe the output of a variety of data mining algorithms and includes probability distributions, predictive models and clusters/clusterings; however, we restrict it here to the sense that it is most commonly used, i.e., in the sense of frequent pattern mining.

A **predictive model M for types T_d, T_c** is a function that takes an object of type T_d and returns one of type T_c , i.e., has the signature $\mathbf{m} :: T_d \rightarrow T_c$. Most often, predictive modelling is concerned with classification, where T_c would be Boolean (for binary classification) or Discrete(S) (for multi-class classification), or regression, where T_c would be Real. In our case, we allow both T_d (description) and T_c (class/target) to be arbitrarily complex data types.

A **probabilistic predictive model P for types T_c, T_d** is a function that takes an object of type T_c and returns a probability distribution over type T_d , i.e., has the signature $\mathbf{p} :: T_c \rightarrow (T_d \rightarrow \mathbf{R}^{0+})$. For discrete T_c , the probability of each possible class value is given by a prediction. For real-valued T_c , the probability distribution can be, for example, assumed to be normal and its mean and standard deviation can be given by the prediction.

A **clustering C on a set of objects S of type T** is a function from S to $\{1, \dots, k\}$, where k is the number of clusters, which has to obey $k \leq |S|$. Unlike all the previously listed types of patterns, a clustering is not necessarily a total function on T, but rather a partial function defined only on objects from

S. Overlapping and soft clusterings, where an element can (partially) belong to more than one cluster have the signature $\mathbf{T} \rightarrow (\{1, \dots, k\} \rightarrow \mathbf{R}^{0+})$.

We can think of C as a matrix B , where $B(e, c)$ states to which degree datum e belongs to cluster c . In conventional clustering $B(e, c) = 1$ if datum e belongs to cluster c and 0 otherwise; only one “1” entry is allowed in each row of B . In overlapping clustering, there can be more than one “1” in each row of B . In soft clustering, the sum of the entries in each row should amount to one.

In predictive clustering, C is a total function on T . In addition, we have $T = (T_d, T_c)$ and we have a predictive model associated with each cluster through a mapping $M :: \{1, \dots, k\} \rightarrow (T_d \rightarrow T_c)$. Performing the function composition of M and C , i.e., applying first C and then M , we get a predictive model on T .

4.3 Data Mining Tasks

In essence, the task of data mining is to produce a generalization from a given set of data. A plethora of data mining tasks has been considered so far in the literature. Here we will focus on four tasks, according to the generalizations produced: approximating the (joint) probability distribution, learning predictive models, clustering and finding valid (frequent) patterns.

Estimating the (Joint) Probability Distribution. A set of data (of type T) is often assumed to be a sample taken from a population according to a probability distribution. A probability distribution/density function assigns a non-negative probability/density to each object of type T . Probably the most general data mining task (Hand et al. 2001) is the task of estimating the (joint) probability distribution D over type T from a set of data items or a sample drawn from that distribution.

As mentioned above, in the most typical case we would have $T = \text{Tuple}(T_1, \dots, T_k)$, where each of T_1, \dots, T_k is Boolean, Discrete(S) or Real. We talk about the joint probability distribution to emphasize the difference to the marginal distributions of each of the variables of type T_1, \dots, T_k : the joint distribution captures the interactions among the variables.

Representing multi-variate distributions is a non-trivial task. Two approaches are commonly used in data mining. In the density-based clustering paradigm, mixtures of multi-variate Gaussian distributions are typically considered (Hand et al. 2001). Probabilistic graphical models, most notably Bayesian networks, represent graphically the (in)dependencies between the variables: Learning their structure and parameters is an important approach to the problem of estimating the joint probability distribution.

Learning a (Probabilistic) Predictive Model. In this task, we are given a dataset that consists of examples of the form (d, c) , where each d is of type T_d and each c is of type T_c . We will refer to d as the description and c as the class or target. To learn a predictive model means to find a mapping from the description to the target, $\mathbf{m} :: T_d \rightarrow T_c$, that fits the data closely. This means that the observed target values and the target values predicted by the model, i.e., c and $\hat{c} = m(d)$, have to match closely.

In the case of learning probabilistic models, we need to find a mapping of the form $\mathbf{m} :: \mathbf{T}_d \rightarrow (\mathbf{T}_c \rightarrow \mathbf{R}^{0+})$. In a more general formulation of the problem, the training examples can have probability distributions over the c values instead of individual c values: This can represent uncertainty in the observations. Few (if any) actual data mining approaches take this stand; most assume that a single value for the target is given for each example.

Many different kinds of predictive models have been considered in data mining. Some examples include classification rules, decision trees and (generalized) linear models. We postpone the discussion on different model kinds (classes) until later in the chapter.

The present task can be viewed as a special case of the task of estimating the probability distribution. If we solve the latter and obtain $P((d, c))$, an estimate of the probability of observing the example (d, c) , we can derive the conditional distribution $P(c|d)$, which is a predictive probabilistic model.

Clustering. Clustering in general is concerned with grouping objects into classes of similar objects (Kaufman and Rousseeuw 1990). Given a set of examples (object descriptions), the task of clustering is to partition these examples into subsets, called clusters. The notion of a distance (or conversely, similarity) is crucial here: examples are considered to be points in a metric space (a space with a distance metric). The goal of clustering is to achieve high similarity between objects within individual clusters (intra-cluster similarity) and low similarity between objects that belong to different clusters (inter-cluster similarity).

In clustering, the examples do not contain a target property to be predicted, but only an object description. Note that a prototype (prototypical example) may be used as a representative for a cluster. This may be, e.g., the mean or the medoid of the examples in the cluster (which are examples with lowest average distance to all the examples in the cluster).

Clustering is known as cluster analysis in statistics, as customer segmentation in marketing and customer relationship management, and as unsupervised learning in machine learning. Conventional clustering focusses on distance-based cluster analysis. In conceptual clustering (Michalski 1980), a symbolic representation of the resulting clusters is produced in addition to the partition into clusters: we can thus consider each cluster to be a concept (much like a class in classification). In predictive clustering (Blockeel et al. 1998), a predictive model is associated with the clustering, so that new instances can be immediately associated with one of the created clusters and its prototype, which can be considered as a representative of the cluster.

In density-based clustering (Hand et al. 2001), clusters correspond to different components of the joint probability distribution, which is assumed to be a mixture model. An example belongs to each cluster to a different degree, determined by the probability that the corresponding component of the mixture assigns to it. In this context, clustering is clearly a special case of the task of estimating the joint probability distribution.

Pattern Discovery. In contrast to the previous three tasks, where the goal is to build a single global model describing the entire set of data given as

input, the task of pattern discovery is to find all local patterns from a given pattern language that satisfy the required conditions. A prototypical instantiation of this task is the task of finding frequent itemsets (sets of items, such as $\{\textit{bread}, \textit{butter}\}$), which are often found together in a transaction (e.g., a market basket) (Aggrawal et al 1993). The condition that a pattern (itemset) has to satisfy in this case is to appear in (hold true for) a sufficiently high proportion (called support and denoted by s) of the transactions in the input dataset.

With the increasing interest in mining complex data, mining frequent patterns is also considered for structured data. We can thus talk about mining frequent subsequences or mining frequent subgraphs in sequence or graph data. We can consider as frequency the multiple occurrences of a pattern in a single data structure (e.g., sequence or graph) or the single occurrences of a pattern in multiple data structures.

The task of finding frequent itemsets (patterns) is typically performed in the context of association analysis (Han and Kamber 2001). After all frequent itemsets are found, one looks for association rules of the form $X \rightarrow Y$, where X and Y are frequent itemsets and the confidence of the rule passes a threshold c . The confidence of the rule $X \rightarrow Y$ is the percentage of transactions containing X that also contain Y . Generalizations of the task of pattern discovery include the discovery of clauses in first order logic (Dehaspe and De Raedt 1997) and the discovery of frequent Datalog queries and query extensions (Dehaspe and Toivonen 1999), the latter being generalizations of finding frequent itemsets and association rules to first order logic.

While the original formulation of the problems of frequent itemset and association rule mining reports all itemsets and rules that pass the support respectively confidence threshold, we can think of these tasks also as ranking tasks, where the itemsets and rules are ordered according to support, respectively confidence. One can then imagine asking for the top- k most frequent itemsets or most confident association rules. This formulation bears an important similarity to the problem of feature ranking and selection, which is often encountered in the context of global (mostly predictive) modelling. Top- k queries of this kind also appear in the context of correlated itemsets and association rules, which can be used for constructing classifiers. In fact, there is an increasing body of research that uses the results of local pattern mining/discovery to build global (predictive) models.

In this context, we can view the tasks of bump hunting (Friedman and Fisher 1999) and subgroup discovery (Kloesgen 2002, Lavrač et al. 2004) as special cases of pattern discovery. Both involve finding groups of examples where the (probability distribution of) the values of a designated target is unusual. Here unusual can be taken to mean unusually large or small, or, more generally, significantly different from the average (or the distribution) over the entire dataset/population. It is in addition desired that these regions be describable in an interpretable form involving simple statements (rules).

Note, finally, that finding frequent patterns can be viewed as a special case of estimating the joint probability distribution. If we think of the joint probability distribution as a surface in multi-dimensional space, then the frequent patterns

(and the groups of examples for which they are true) would correspond to peaks of the surface. The threshold on the frequency would correspond to a hyperplane that would cut off patterns below the given frequency.

5 The Dual Nature of Patterns and Models

Patterns and models inherently have a dual nature. According to the definitions from the previous sections, they are functions that take as input data points and map them to probabilities, Booleans, class predictions or probabilities thereover, or cluster assignments. On the other hand, they can be treated as data structures and as such represented, stored and manipulated.

Let us illustrate this with a simple example. Suppose we have a frequent itemset consisting of the items bread and butter. We can view this as a set, namely $\{bread, butter\}$, and store it in a database. In this fashion, we can store the frequent itemsets derived from a set of transactions. On the other hand, from the functional viewpoint, the itemset represents a mapping from transactions to Booleans. The transactions which contain the itemset, i.e., both bread and butter, are assigned the value true, i.e., the pattern holds true for such transactions. For example, the transaction $\{bread, butter, milk\}$ subsumes our itemset and yields the value true, while $\{beer, peanuts, butter\}$ does not and yields the value false.

5.1 The Data Aspect: Classes of Patterns and Models

Many different kinds of predictive models have been considered in the data mining literature. Classification rules, decision trees and linear models are just a few examples. We will refer to these as model classes. In the case of patterns we will talk about pattern classes.

A class of patterns C_P on type T is a set of patterns P on type T , expressed in a language L_P . Similarly, a class of models C_M on types T_d, T_c is a set of models M on types T_d, T_c , expressed in a language L_M . In the same fashion, we can define classes of probability distributions C_D and clusterings C_C .

The languages $L_P/L_M/L_D/L_C$ refer to the data part of the patterns and models. They essentially define data types for representing the patterns and models. For example, if we have data types $T_d = (\text{Real}, \text{Real})$ and $T_e = \text{Real}$, linear models would be represented by three real-valued coefficients and would need a data type $T_l = (\text{Real}, \text{Real}, \text{Real})$ to be represented.

Suppose we have a dataset where data items correspond to descriptions of individuals, each individual being described by a tuple of the form (Gender, Age, HairColor), where Gender = Discrete($\{M, F\}$), Age = Real, HairColor = Discrete($\{Blond, Brown, Black, Red, Other\}$), and the target is of type Education = Discrete($\{None, Elementary, High, College, BSc, MSc, PhD\}$). The language of decision trees for this case would be the language of tree structures with tests like HairColor=Blond in the internal nodes and predictions like Education=PhD in the leaves. The elements of this language (its alphabet) depend on the attributes and their values, and vary with the underlying data type.

5.2 The Function Aspect: Interpreters

There is usually a unique mapping from the data part of a pattern/model to the function part. This takes the data part of a pattern/model as input, and returns the corresponding function as an output. The mapping we refer to is inherently second/higher order (Lloyd 2003) since it has a function as an output.

This mapping can be realized through a so-called interpreter. An interpreter takes as input (the data part of) a pattern and an example, and returns the result of applying the (function part) of the pattern to the example. Given a data type d , an example E of type d , and a pattern P of type $\mathbf{p} :: \mathbf{d} \rightarrow \mathbf{bool}$, an interpreter I returns the result of applying P to E , i.e., $I(P, E) = P(E)$.

The signature of the interpreter is $\mathbf{i} :: \mathbf{p} \rightarrow \mathbf{d} \rightarrow \mathbf{bool}$. If we apply the interpreter to a pattern and an example, we obtain a Boolean value. In functional programming (Thompson 1999), we can evaluate the interpreter only partially, i.e., apply it only to the data part of a pattern, obtaining as a result the function part of the pattern. The partial evaluation $\mathbf{i} \mathbf{p}$ has a signature $\mathbf{d} \rightarrow \mathbf{bool}$.

The interpreters map from the data part of a pattern/model to the function part. Suppose we are given a linear model with coefficients a , b , and c . The interpreter of linear models I_l would, given a , b , and c , and a data tuple of the form (x, y) , return the value of the linear combination $ax + by + c$. A partial evaluation/application of the interpreter to the tuple of constant coefficients of the linear model $I_l(a, b, c)$ would yield the linear function $aX + bY + c$: This linear function can then be applied to specific tuples (x, y) to yield predictions.

The interpreter is crucial for the semantics of a class patterns/models: a class of patterns/models is only completely defined when the corresponding interpreter is defined (e.g., I_P/I_M for patterns/models are parts of the definition of the class C_P/C_M). To illustrate this, consider rule sets, which may be ordered or unordered. Both can actually be represented by the same list of rules: It is the interpreter that treats the rules as ordered or unordered. In the first case, the rules are considered in the order they appear in the list and the first rule that applies to a given example is taken to make a prediction. In the second case, all rules from the list that apply to a given example are taken, and their predictions combined to obtain a final prediction.

6 Constraints in Data Mining: Introduction

Let us recall briefly that data mining is concerned with finding patterns/models that are valid in a given set of data. The key ingredients of data mining thus include data, data mining tasks, and patterns/models, which we have elaborated on in some detail in the previous sections. We now turn to the issue of pattern/model validity. Essentially, we say that a pattern is valid if it satisfies a given set of constraints. The constraints considered depend heavily on the data mining task at hand, and so does the concept of validity. In this section, we introduce the notion of constraints and discuss the different types of constraints.

A view generally held is that constraints are Boolean functions on patterns/models. A constraint is either satisfied or not satisfied. Given that patterns and

models have a dual nature, i.e., have both a data and a function aspect, we can have constraints on each of these aspects.

6.1 Language Constraints

Language constraints concern the data part of a pattern/model. Boolean language constraints define a subclass/sublanguage of the class of patterns/models considered. For example, in the context of mining frequent itemsets, we might be interested only in itemsets where a specific item, e.g., *beer* occurs. Or, in the context of learning predictive models, we may be interested only in decision trees that have a specific attribute in the root node and, in addition, do not have more than seven leaves.

These are language constraints and refer to the data part only. We can check whether they are satisfied or not without accessing the data that we have been given as a part of the data mining task. If we are in the context of inductive databases and queries, queries on the data part of patterns/models are composed of primitive language constraints.

Language constraints may also involve (cost) functions on the data part of patterns/models. An example of these is the size of a decision tree, mentioned above. Another example would be the cost of an itemset (market basket), in the context where each item has a price. The cost functions as discussed here are mappings from the data part of a pattern/model to non-negative reals and Boolean language constraints can put thresholds on the values of these functions.

6.2 Evaluation Constraints

Evaluation constraints correspond to inductive queries that concern the function aspect of patterns/models. Evaluation constraints are typically Boolean functions, i.e., statements, involving evaluation functions and comparing them to constant thresholds. Evaluation functions measure the validity of patterns/models on a given set of data.

Evaluation functions are functionals, i.e., they take a function (in this case a pattern or a model) as input and return a scalar (real) value as output. The set of data is an additional input to the evaluation functions. For example, the frequency of a pattern on a given dataset is a typical evaluation function. Similarly, the classification error of a predictive model is also an evaluation function. Evaluation constraints typically compare the value of an evaluation function to a constant threshold, e.g., minimum support or maximum error.

Constraints on the function part of a pattern/model may also involve some general property of the function, which does not depend on the specific dataset considered. For example, we may only consider functions that are convex or symmetric or monotonic in certain variables. These properties are usually defined over the entire domain of the function, i.e., the corresponding data type, but may be checked for the specific dataset at hand.

6.3 Optimization Constraints

Many Boolean constraints are obtained by imposing a threshold on the value of a function(al). This can be a threshold on a cost function over the data part of a pattern/model or on an evaluation function(al) on the function part of the model. Boolean constraints are either satisfied or not.

On the other hand, optimization constraints ask for (a fixed-size set of) patterns/models that have a maximal/minimal value for a given cost or evaluation function. Example queries involving such constraints would ask for the k most frequent itemsets or the top k correlated patterns. Alternatively, we might ask for the most accurate decision tree of size five, or the smallest decision tree with classification accuracy of at least 90%.

In this context, optima for the cost/evaluation function at hand are searched for over the entire class of patterns/models considered, in the case the optimization constraint is the only one given. But, as illustrated above, optimization constraints often appear in conjunction with (language or evaluation) Boolean constraints. In this case, optima are searched for over the patterns/models that satisfy the given Boolean constraints.

6.4 Soft Constraints

If we define language and evaluation constraints as Boolean functions, we view them as hard constraints. A constraint is either satisfied or not satisfied by a pattern. The fact that constraints actually define what patterns are valid or interesting in data mining, and that interestingness is not a dichotomy (Bistarelli and Bonchi 2005), has lead to the introduction of so-called soft constraints.

Instead of dismissing a pattern for violating a constraint, we might consider the pattern incurring a penalty for violating a constraint. In the cases where we typically consider a larger number of binary constraints, such as must-link and cannot-link constraints in constrained clustering (Wagstaff and Cardie 2000), a fixed penalty may be assigned for violating each constraint. In case we are dealing with evaluation constraints that compare an evaluation function to a threshold, the penalty incurred by violating the constraint may depend on how badly the constraint is violated. For example, if we have a size threshold of five, and the actual size is six, a smaller penalty would be incurred as compared to the case where the actual size is twenty.

In the hard constraint setting, a pattern/model is either a solution or not. In the soft constraint setting, all patterns/models are solutions to a different degree. Patterns that satisfy the constraint(s) get zero penalty: This leads to an optimization problem where we look for patterns with minimum penalty.

6.5 The Task(s) of (Constraint-Based) Data Mining

Having set the scene, we can now attempt to formulate a very general version of the problem addressed by data mining. We are given a dataset D , drawn according to some probability distribution P , consisting of objects of type T . We

are also given a data mining task, one of the four listed in a previous section (estimating the probability distribution P , learning a predictive model, clustering or pattern discovery). We are further given a class of generalizations C_G (patterns/models/clustering/probability distributions), from which to find solutions to the data mining task at hand. Finally, a set of constraints C is given, which can include both language and evaluation constraints.

The problem addressed by constraint-based data mining is to find a set of generalizations G from C_G that satisfy the constraints in C , if C is boolean, or optimize the constraints in C , if C contains optimization or soft constraints. A desired cardinality on the solution set is usually specified.

In the above formulation, all of data mining is really constraint-based data mining. We argue that the ‘classical’ formulations of and approaches to data mining tasks, such as clustering and predictive modelling, are a special case of the above formulation. A major difference between the ‘classical’ data mining paradigm and the ‘modern’ constraint-based one is that the former typically consider only one optimization constraint, such as minimize predictive error or intra-cluster variance, and requires only one solution (predictive model or clustering).

A related difference concerns the fact that most of the ‘classical’ approaches to data mining are heuristic and do not give any guarantees regarding the solutions. For example, a decision tree generated by a learning algorithm is typically not guaranteed to be the smallest or most accurate tree for the given dataset. On the other hand, constraint-based mining approaches have typically been concerned with the development of so-called ‘optimal solvers’, i.e., data mining algorithms that return the complete set of solutions that satisfy a given set of constraints or the truly optimal solutions (e.g., the k itemsets with highest correlation to a given target) in the context of optimization constraints.

7 The Key Ingredients of Data Mining Algorithms

7.1 Generality and Refinement Operators

The notion of generality is a key notion in data mining, in particular for the task of pattern discovery. To find patterns/models valid in the data, data mining algorithms search the space of patterns defined by the class of patterns/models considered, possibly additionally restricted by language constraints. To make the search efficient, the space of patterns/models is typically ordered by a generality or subsumption relation. A generality relation on a set (of patterns/models) is a partial order on that set.

The generality relation typically refers to the function part of a pattern/model. The corresponding notion for the data part is that of refinement. A typical example of a refinement relation is the subset relation on the space of itemsets. This relation is a partial order on itemsets and structures itemsets into a lattice structure, which is typically explored during the search for, e.g., frequent itemsets. The refinement relation is typically the closure of a refinement operator, which performs minimal refinements. In the case of itemsets, it takes an itemset

and adds an item to it: if all possible items are *beer*, *diapers*, *milk* and *peanuts*, the refinements of the itemset $i_1 = \text{beer}$ are $i_2 = \text{beer, diapers}$, $i_3 = \text{beer, milk}$ and $i_4 = \text{beer, peanuts}$. Starting with the empty itemset, we can obtain any itemset through a sequence of refinements (applications of the refinement operator).

We can think of refinement and generality as expressing the same relation between patterns at the data (or syntax) level and the function (or semantics) level. In logic, we talk about subsumption in the first case and logical entailment (implication) in the second (Džeroski 2007). For example, if we take the itemsets from the above paragraph, i_2 is a refinement of i_1 at the data level means $i_1 \subseteq i_2$. At the function level, i_1 and i_2 are Boolean functions over transactions, $i_1(t)$ and $i_2(t)$. Generality here has the following meaning: i_1 is more general than i_2 at the function level means $\forall t : i_2(t) \models i_1(t)$, where \models denotes logical implication.

In the ideal case, the notions of refinement at the syntactic and generality at the semantic level (resp. the data and function level) coincide. Whether this is actually the case depends on the interpreter for the class of patterns considered. These issues have received considerable attention in the area of inductive logic programming (Lavrač and Džeroski 1994, Džeroski 2007), where both data and patterns are represented in first order logic. The notions of generality and refinements are also directly relevant to and ubiquitously used in mining predictive models and other forms of generalizations. However, the notion of semantic generality does not transfer in a straightforward manner to the case of functions that are not binary/Boolean (i.e., to clusterings, probability distributions and predictive models in general).

7.2 Distances and Prototypes

Distance functions are of crucial importance for the design of many data mining algorithms, most notably for clustering and predictive modelling. A distance function d for type T is a mapping from pairs of objects of type T to non-negative reals: $d :: \mathbf{T} \times \mathbf{T} \rightarrow \mathbf{R}^{0+}$. A distance function has to satisfy three properties: (1) $d(x, y) \geq 0$, (2) $d(x, y) = 0$ if and only if $x = y$, and (3) $d(x, y) = d(y, x)$.

Note that the distance between two objects should be zero if and only if the two objects are identical (this property is called discernibility) and that a distance function should be symmetric. Properties (1) and (2) taken together produce positive definiteness. An additional property of interest for distance functions is the triangle inequality: (4) $d(x, z) \leq d(x, y) + d(y, z)$. A distance function that satisfies the triangle inequality is called a metric.

While it is immediately obvious that we need distances for distance-based clustering (where by definition we want to minimize the distance between objects in a cluster), it is may be less obvious why we need them for predictive modelling. The primary reason is the need to assess the predictions of a model: we need to compare the true value of the target to the predicted one, for any given example. In most predictive modelling approaches, it is assumed that the error/penalty incurred by predicting x instead of y is the same as the one incurred by predicting y instead of x , and equal to some distance function $d(x, y)$.

For any type of data we can easily define the distance function δ , which takes the value zero for pairs of identical data points and one for all other pairs: $\delta(x, x) = 0$ and $\delta(x, y) = 1$ for $x \neq y$. In fact, this is the distance function most commonly used for discrete/nominal data types in data mining algorithms. For real numbers, we can use $|x - y|$ as the distance between x and y .

Related to the notion of distance is the notion of a prototype. A prototype is something that is representative of a category of things, in this case of all the objects in a given set S . The prototype of a set of objects (defined in the context of a given distance d) is the object o that has the lowest average square distance to all of the objects in S : $o = \operatorname{argmin}_q \sum_{X \in S} d^2(X, q)$. Note that the quantity that we want to minimize in this formula is a generalization of the notion of variance from a set of real numbers to a set of arbitrary objects.

A prototype function p for objects of type T , takes as input a set S of objects of type T , and returns an object of type T , i.e., the prototype: $p :: \text{Set}(\mathbf{T}) \rightarrow \mathbf{T}$. We can consider two possibilities here: (a) the prototype is an arbitrary object of type T or (b) the prototype is one of the objects from S . In the case (b), the prototype can be computed with $|S|^2$ distance computations by substituting q with each of the objects in s . In the case (a), the space of candidate prototypes may easily be infinite. We thus need to have a closed algebraic form of the prototype or should resort to approximative algorithms to compute it.

In vector spaces, such as the Euclidean spaces \mathbf{R}^n , where objects may be scaled or added, the prototype of a set of objects can be defined in closed form as the centroid of the set. The notion of centroid generalizes the notion of mean from sets of real numbers to multidimensional spaces. The centroid is defined as the (weighted) mean / average of the vectors in the set: by default each vector has an equal weight ($1/|S|$), although different weights may be assigned to different vectors. For example, given a set S of vectors x_i in the Euclidean space \mathbf{R}^n , each of the form $x_i = (x_{i1}, \dots, x_{ik})$, the centroid \bar{x} is defined as $\bar{x} = (\bar{x}_1, \dots, \bar{x}_i)$, where $\bar{x}_j = \sum_{i=1}^{|S|} x_{ij} / |S|$. The centroid can be computed by $|S|$ addition computations and one scaling computation.

Prototypes and prototype functions are directly relevant to the clustering task of data mining, as well as the task of predictive modelling. Quite often, a prototype is associated with each cluster. In predictive modelling approaches which partition the space of training examples, such as tree-based and rule-based methods, the prediction of a rule/tree leaf is typically obtained by constructing a prototype of the (target part) of the examples covered by the rule/leaf.

7.3 Features and Background Knowledge

The term ‘feature’ is heavily used in pattern recognition (Bishop 2006), where features are individual measurable properties of a phenomena or object being observed. Features are usually numeric, but structural features (such as strings and graphs) are used in syntactic pattern recognition. While different areas of pattern recognition (such as image analysis or speech recognition) obviously use different features, once the features are decided upon, a relatively small set of

algorithms is used to analyze the resulting data table. These algorithms include, e.g., linear discriminants / regression and probabilistic (naive Bayes) approaches.

Some approaches to data mining do not explicitly rely on a feature-based representation of the data analyzed. For example, many distance-based approaches to prediction (e.g., nearest neighbor methods) and clustering (e.g., hierarchical agglomerative/divisive clustering or k -medoids) only need a distance function on the underlying data (type). However, even for these, the distances are most commonly calculated through a set of features. The majority of data mining algorithms, though, crucially depend on the use of features (linear regression, naive Bayes, decision trees, classification rules, to name the most common ones).

Defining an appropriate set of features for a data mining problem at hand is still much of an art. However, it is also a step of key importance for the successful use of data mining. In the following, we try to formalize the notion of a feature and briefly discuss principled approaches to generating features.

Suppose d is a datum (structured object) of type T . Note that d can be, e.g., an image represented by an array of real numbers, or a recording of speech, represented by a sequence of real numbers. A feature f of objects of type T is a mapping from objects of type T to a primitive data type (Boolean, Discrete or Real) and $f(d)$ refers to the value of the feature for the specific object d .

There are at least three ways to arrive at features for a given object d of type T . First, the feature may have been directly observed and thus be a part of the representation of d . For example, if we have a molecule represented by its molecular weight, hydrophobicity and activity against a given species of bacteria, hydrophobicity as a bulk property is typically measured directly and is a feature of the molecule.

The other two ways are related to background knowledge concerning the structure of the object or concerning domain knowledge. Suppose molecules are represented by labeled graphs with vertices representing atoms and edges representing bonds between atoms: atoms would be labeled by the type of element (e.g., nitrogen, oxygen) and edges would be labeled by the type of bond (e.g., single, double, triple). In this context, a (simple) structural feature might indicate the presence/absence of a carbon and oxygen atom connected by a double bond (a $C = O$ group) in a given molecule. To illustrate how a feature may be derived through the use of some domain knowledge, consider molecules again. The presence of (complex) structures, such as certain functional groups (alcohols) or complexes thereof (triple fused rings) might be used as features. Connectivity indices calculated on the entire graph might also be used as features.

Background knowledge can be thought of as a set of mappings that generate new features, either directly, as in the case of connectivity indices on graphs mentioned above, or indirectly. In the second case, a mapping from the background knowledge would map an object from one representation to another, and features can be generated from the latter. For images, such a mapping might perform image segmentation and describe each segment with a new set of features, thus transforming the learning problem to a completely different representation.

From the latter, new features can then be generated directly or through the further use of domain knowledge.

7.4 Kernels

Kernel Methods (KMs, Shawe-Taylor and Cristianini 2004) in general, and Support Vector Machines (SVMs) in particular, are among the most successful recent developments within the machine learning and data mining communities. KMs can be used to address different tasks of data mining, such as clustering, classification, and regression, for general types of data, such as sequences, text documents, sets of points, vectors, images, etc. KMs (implicitly) map the data from its original representation into a high dimensional feature space, where each coordinate corresponds to one feature of the data items, transforming the data into a set of points in a Euclidean / linear space. Linear analysis methods are then applied (such as separating two classes by a hyperplane), but since the mapping can be nonlinear, nonlinear concepts can effectively be captured.

Technically, a kernel k corresponds to the inner product in some feature space. The computational attractiveness of kernel methods comes from the fact that quite often a closed form of these feature space inner products exists. The kernel can then be calculated directly, thus performing the feature transformation only implicitly without ever computing the coordinates of the data in the ‘feature space’. This is called the kernel trick.

Whether, for a given function k with signature $k :: \mathbf{T} \times \mathbf{T} \rightarrow \mathbf{R}$, a feature transformation ϕ exists from \mathbf{T} to a Hilbert space H , such that $k(x, x') = \langle \phi(x), \phi(x') \rangle$ for all x, x' can be checked by verifying that the function is positive definite. A symmetric function k on pairs of data points of type T is a positive definite kernel on T if, for all positive integers n , $x_1, \dots, x_n \in T$ and $c_1, \dots, c_n \in \mathbf{R}$, it holds that $\sum_{i,j \in 1, \dots, n} c_i c_j k(x_i, x_j) \geq 0$. While it is not always easy to prove positive definiteness for a given kernel, positive definite kernels do have nice closure properties. In particular, they are closed under sum, direct sum, multiplication by a scalar, product, tensor product, zero extension, pointwise limits, and exponentiation (Shawe-Taylor and Cristianini 2004).

Probably the simplest kernel is the linear one, defined for tuples of real numbers (x, x') as $k(x, x') = \langle x, x' \rangle$, where $\phi(x) = x$. If we have $x = (x_1, x_2)$ and $x' = (x'_1, x'_2)$, then $k(x, x') = x_1 x'_1 + x_2 x'_2$. Other kernels include the polynomial $k(x, x') = (\langle x, x' \rangle + 1)^p$ and the exponential $k(x, x') = e^{-\gamma \|x - x'\|^2}$ kernel.

At the conceptual level, kernels elegantly relate to both features and distances. As mentioned above, a kernel k (implicitly) defines a mapping from the original space to a Hilbert space $x \rightarrow \phi(x)$, the latter being the feature space implicitly associated with the kernel. A (pos. def.) kernel also defines a distance: if k is a kernel, then $d(x, y) = \sqrt{k(x, x) - 2k(x, y) + k(y, y)}$ is a (pseudo)metric.

At the practical level, kernel functions have been introduced for different types of data, such as vectors, text, and images, including structured data, such as sequences and graphs (Gaertner 2003). There are also many algorithms capable of operating with kernels: these include SVMs, Fisher’s linear discriminant analysis

(LDA), principal components analysis (PCA), ridge regression, spectral clustering, and many others. Since any kernel can be used with any kernel-algorithm, it is possible to construct many combinations, such as regression over DNA sequences, classification of documents, and clustering of images.

8 Constraints in Data Mining: Revisited

8.1 Evaluation Functions for the Basic Data Mining Tasks

The evaluation functions used in evaluation constraints are tightly coupled with the data mining task at hand. If we are solving a predictive modelling problem, the evaluation function used will most likely concern predictive error. If we are solving a frequent pattern mining problem, the evaluation function used will definitely concern the frequency of the patterns.

For **predictive models** with a signature $\mathbf{m} :: \mathbf{T}_d \rightarrow \mathbf{T}_c$, we need a distance (or cost) function d_c on objects of type T_c to define the notion of predictive error. For a given model m and a dataset D , the average predictive error of the model is defined as $1/|D| \times \sum_{e=(a,t) \in D} d_c(t, m(a))$. For each example $e = (a, t)$ in the dataset, which consists of a descriptive (attribute) part a and target (class) part t , the prediction of the model $m(a)$ is obtained and its distance to the true class value t is calculated. Analogously, the notion of mean squared error would be defined as $1/|D| \times \sum_{e=(a,t) \in D} d_c^2(t, m(a))$.

The notion of cost-sensitive prediction has been recently gaining increasing amounts of attention in the data mining community. In this setting, the errors incurred by predicting x instead of y and predicting y instead of x , are typically not the same. The corresponding misprediction (analogous to misclassification) cost function is thus not symmetric, i.e., is not a distance. The notion of average misprediction cost can be defined as above, where the distance $d(x, y)$ is replaced by a cost function $c(x, y)$.

In the case of **probabilistic models**, which predict a probability distribution over the target type and are of the form $\mathbf{m} :: \mathbf{T}_d \rightarrow (\mathbf{T}_c \rightarrow \mathbf{R}^{0+})$, we need a distance function on probability distributions over the target data type T_c . When T_c is discrete and takes values from $S = \{s_1, \dots, s_k\}$, we can represent a probability distribution on T_c with a vector $p = (p_1, \dots, p_k)$ of probabilities of its possible values. We can then use distances on vectors of reals. However, distances or cost functions that explicitly take into account the fact that we are dealing with probability distributions can also be taken, such as the likelihood-ratio defined for two distributions p and q as $\sum_{i=1}^k p_i \log(p_i/q_i)$. The latter is a special case of the Kullback-Leibler divergence, defined also for probability distributions / densities over continuous variables.

For the task of **estimating the probability distribution** of objects of type T , we need a scoring function for distributions / densities. The most commonly used ones are based on likelihood or log-likelihood (Hand et al. 2001). Given a dataset D and a probability distribution p , the likelihood function is defined as $L(p) = \prod_{e \in D} p(e)$ and the log-likelihood function as $\log L(P) = \sum_{e \in D} \log p(e)$.

Another possibility for evaluating a candidate probability distribution p is to calculate the integrated (average) squared error between p and the true distribution p^* . This is defined as $\int_x (p(x) - p^*(x))^2 dx$. We can ignore terms that do not depend on p , yielding $\int_x p^2(x) dx - \int_x p^*(x)p(x) dx = \int_x p^2(x) dx - E(p(x))$, where each of the terms can be approximated to obtain an estimate of the true integrated (average) squared error for p : $E(p(x))$ denotes the expectation of $p(x)$.

Density-based **clustering** is a direct special case of the task of estimating the joint probability distribution, where clusters correspond to different components of the joint probability distribution, which is assumed to be a mixture model. A mixture model has the form $p(x) = \sum_{i=1}^k \pi_k p_k(x)$ and decomposes the overall density (or distribution) for x into a weighted linear combination of k component or class densities. In this case, the same evaluation functions as for the task of estimating the probability distribution can be applied.

For the traditional partition-based clustering approach, the quality of a clustering is typically evaluated with intra-cluster variance (ICV). For a clustering with k clusters C_i with $D = \cup_{i=1}^k C_i$, we have $ICV = \frac{1}{|D|} \sum_{i=1}^k |C_i| Var(C_i)$, where C_i is the set of elements of cluster i . $Var(C_i)$ is the intra-cluster variance of cluster i and is defined as $Var(C_i) = \sum_{e \in C_i} d^2(e, \bar{C}_i)$, where \bar{C}_i is the prototype of cluster C_i with respect to the distance d .

Finally, for the task of **pattern discovery**, with the discovery of frequent patterns as the prototypical instantiation, the primary evaluation function is frequency. Recall that patterns are Boolean functions and have the signature $\mathbf{p} :: \mathbf{T} \rightarrow \mathbf{bool}$. For a dataset D of objects of type T , the frequency of a pattern p is defined as $f(p, D) = |\{e | e \in D, p(e) = true\}|$.

8.2 Cost Functions for Language Constraints

The cost functions that are used in language constraints concern the data part of generalizations (patterns/models/...). Most often, these functions are related to the size/complexity of the generalizations. They are different for different classes of generalizations, e.g., for itemsets, mixture models of Gaussians, linear models or decision trees. For itemsets, the size is the cardinality of the itemset, i.e., the number of items in it. For decision trees, it can be the total number of nodes, the number of leaves or the depth of the tree. For linear models, it can be the number of variables (with non-zero coefficients) included in the model.

More general versions of cost functions involve costs of the individual language elements, such as items or attributes, and sum/aggregate these over all elements appearing in the pattern/model. These are motivated by practical considerations, e.g., costs for items in an itemset and total cost of a market basket. In the context of predictive models, e.g., attribute-value decision trees, it makes sense to talk about prediction cost, defined as the total cost of all attributes used by the model. For example, in medical applications where the attributes correspond to expensive lab tests, it might be useful to upper-bound the prediction cost of a decision tree.

Language constraints as commonly used in constraint-based data mining involve thresholds on the values of cost functions (e.g., find a decision tree of size

at most ten leaves). They are typically combined with evaluation constraints, be it threshold or optimization (e.g., find a tree of size at most 10 with classification error of at most 10% or find a tree of size at most 10 and the smallest classification error). Also, optimization constraints may involve the language-related cost functions, e.g., find the smallest decision tree with classification error lower than 10%.

In the ‘classical’ formulations of and approaches to data mining tasks, scoring functions often combine evaluation functions and language cost functions. The typical score function is a linear combination of the two, i.e., $Score(G, D) = w_E \times Evaluation(G, function, D) + w_L \times LanguageCost(G, data)$, where G is the generalization (pattern/model) scored and D is the underlying dataset. For predictive modelling, this can translate to $Score = w_E \times Error + w_S \times Size$.

8.3 Monotonicity and Closedness

The notion of monotonicity of an evaluation (or cost) function on a class of generalizations is often considered in constraint-based data mining. In mathematics, a function $f(x)$ is monotonic (monotonically increasing) if $\forall x, y : x < y \rightarrow f(x) \leq f(y)$, i.e., the function preserves the $<$ order. If the function reverses the order, i.e., $\forall x, y : x < y \rightarrow f(x) \geq f(y)$, we call it monotonically decreasing.

In data mining, in addition to the order on Real numbers, we also have a generality order on the class of generalizations. The latter is typically induced by a refinement operator. We say that $g_1 \leq_{ref} g_2$ if g_2 can be obtained from g_1 through a sequence of refinements (and thus g_1 is more general than g_2): we will refer to this order as the refinement order.

An evaluation (or cost) function is called monotonic if it preserves the refinement order or anti-monotonic if it reverses it. More precisely, an evaluation function f is called monotonic if $\forall g_1, g_2 : g_1 \leq_{ref} g_2 \rightarrow f(g_1) \leq f(g_2)$ and anti-monotonic (or monotonically decreasing) if $\forall g_1, g_2 : g_1 \leq_{ref} g_2 \rightarrow f(g_1) \geq f(g_2)$.

Note that the above notions are defined for both evaluation/cost functions that refer to the function part of a generalization and for functions that refer to the data part. In this context, the frequency of itemsets is anti-monotonic (it decreases monotonically with the refinement order). The total cost of an itemset and the total prediction cost of a decision tree, on the other hand, are monotonic.

In the constraint-based data mining literature (Boulicaut and Jeudy 2006), the refinement order considered is typically the subset relation on itemsets (\leq_{ref} is identical to \subseteq). A constraint C (taken as a Boolean function) is considered monotonic if $i_1 \leq_{ref} i_2 \wedge C(i_1)$ implies $C(i_2)$. A maximum frequency constraint of the form $freq(i) \leq \theta$, where θ is a constant, is monotonic. Similarly, minimum frequency/support constraints of the form $freq(i) \geq \theta$, the ones most commonly considered in data mining, are anti-monotonic. A disjunction or a conjunction of anti-monotonic constraints is an anti-monotonic constraint. The negation of a monotonic constraint is anti-monotonic and vice versa.

The notions of monotonicity and anti-monotonicity are important because they allow for the design of efficient constraint-based data mining algorithms. Anti-monotonicity means that when a pattern does not satisfy a constraint C ,

then none of its refinements can satisfy C. It thus becomes possible to prune huge parts of the search space which can not contain interesting patterns. This has been studied within the learning as search framework (Mitchell, 1982) and the generic levelwise algorithm from (Mannila and Toivonen, 1997) has inspired many algorithmic developments.

Finally, let us mention the notion of closedness. A pattern (generalization) is closed, with respect to a given refinement operator \leq_{ref} and evaluation function f , if refining the pattern in any way decreases the value of the evaluation function. More precisely, x is closed if $\forall y, x \leq_{ref} y : f(y) < f(x)$. While this notion has primarily been considered in the context of mining frequent itemsets, where it plays an important role in condensed representations (Calders et al. 2005), it can be defined analogously for other types of patterns, as indicated above.

8.4 Multi-objective Optimization and Constraint-Based Data Mining

The moment we consider more than one evaluation or cost function in the context of a single data mining task, we are dealing with a multi-objective optimization problem. In multi-objective optimization we wish to simultaneously optimize several (possibly conflicting) objectives. More precisely, we want to minimize each component of a vector of objective/evaluation functions $\mathbf{f} = (f_1, \dots, f_m)$ simultaneously.

In the context of multi-objective optimization, the notions of Pareto dominance and Pareto optimality are important. We say that a vector of values of the objective functions g weakly dominates another vector h iff $g_i \leq h_i$ for all i . If, in addition, $g_j < h_j$ for at least one j , we say that g dominates h . If $g_i < h_i$ for all i , we are talking about strict Pareto dominance.

The weak Pareto dominance is a natural generalization of the \leq relation on real numbers. While \leq induces a total order on reals, the weak Pareto dominance induces only a partial order on vectors of reals. This means that two objective vectors (and therefore two solutions) can be incomparable: In case of conflicting objectives the multi-objective optimization problem can have multiple optimal solutions. A solution and its corresponding vector of objective function values are Pareto optimal if they are not Pareto dominated by any other solution/vector in the space considered. All Pareto optimal solutions compose the Pareto optimal set, while the corresponding objective vectors constitute the Pareto optimal front.

The typical approach to multi-objective optimization taken within the data mining community, as indicated earlier in this section, is to transform the multi-objective optimization problem into a single-objective problem, by combining the individual objectives using a weighted sum. A single solution (e.g., predictive model) is then acquired by solving the corresponding single-objective problem (e.g., optimizing the weighted sum of error and complexity of the predictive model). This approach is recognized in multi-objective optimization as an application of the ‘preference-based principle’ (Deb 2001), where the user explicitly specifies her preference for the different objectives (e.g., weights) in advance.

In contrast, using the ‘ideal principle’, the multi-objective problem is first solved and only then the user selects a single solution among several alternatives, using preference information. The ideal principle is ideal in the sense that it does not demand from the user to set a preference for the objectives before optimization. Only when several tradeoff solutions are known, the user chooses the preferred one among them.

The ideal principle for multi-objective optimization seems very suitable in the context of inductive databases/queries and constrained based data mining, where multiple objectives are often employed and multiple solutions are usually expected. After a set of solutions has been obtained, this can be the input for further (inductive) queries that would allow the user to express her preference and select a solution. Unfortunately, few approaches in data mining and machine learning employ the ideal approach, with the notable exception of Tušar (2007). This leaves ample space for the development of truly multi-objective approaches to constraint-based data mining and inductive querying, which we believe is a promising direction for further research.

9 Generic Algorithms for Mining Structured Data

Early on in this article, we have stated that a unifying approach to mining different types of data would be a significant step towards a general data mining framework. Having elaborated on the different types of data, different data mining tasks and the basic components of data mining algorithms, we can now outline what such a unifying approach would look like. Essentially, it should provide an elegant mechanism for defining the key ingredients of data mining algorithms, such as generality/refinement operators, distances, features and kernels, for the different types of data considered.

The key data mining notions have been studied extensively and are reasonably well understood for primitive data types. The basic idea of the unified approach to mining structured data is to derive the key components of data mining algorithms for a complex data type (built through using type constructors) from information on the structure of that type (what constructors on what simpler data types) and the key components for the simpler data types. For example, a distance function d on tuples of type $\text{Tuple}(T_1, \dots, T_n)$ can be composed from distance functions d_i on types T_i by adding up the distances for each tuple component $d(x, y) = d((x_1, \dots, x_n), (y_1, \dots, y_n)) = \sum_{i=1}^n d_i(x_i, y_i)$.

Note that there are several degrees of freedom when constructing the distance for a more complex data type from distances on simpler types. One dimension is the type constructor, which besides $\text{Tuple}()$ may also be $\text{Set}()$ or $\text{Sequence}()$. Even if we fix the type constructor to $\text{Tuple}()$, there are alternative ways to combine the distances on the simpler types: for example, we can take the square root of the sum of squared distances for each tuple component $d(x, y) = \sqrt{\sum_{i=1}^n d_i^2(x_i, y_i)}$ instead of the above. Finally, different distances may exist for the same simpler type (e.g., d_{11} and d_{12} on type T_1 , instead of just d_1).

The approach we have outlined above for distances can be also applied for generality/refinement operators, features and kernels. In the remainder of this section, we outline how this would be done. We also discuss generic data mining algorithms that would work on arbitrary types of data: These would be parameterized with the key components mentioned above.

9.1 Distances and Distance-Based Algorithms

Distances. We have already discussed how distances for simpler data types can be combined to derive distances for more complex data types created with the `Tuple()` constructor: this is done in a straightforward fashion by adding up the distances along each component. For the `Set()` constructor, the situation is more complicated. Many proposals exist in the literature for constructing a distance on sets of objects of type T from a distance on objects of type T . A concise, yet comprehensive, overview is given by Kalousis et al. (2007).

The simpler option of constructing a distance on sets is to calculate the distances for all pairs of elements $D(A, B) = \{d(a_i, b_j) | (a_i, b_j) \in A \times B\}$, then aggregate this $d(A, B) = f(D(A, B))$: Here A and B are sets, a_i/b_j are elements thereof, and f an aggregation function. The three options of $f = \min, \max$ or *average* give rise to the so-called single, complete and average linkage, which are in common use, e.g., in hierarchical clustering. Even if $d(a, b)$ is a metric, none of the three variants here is a metric and only single linkage is a true distance function, as the other two are not reflexive. The Hausdorff distance, defined as $d_H(A, B) = \max(\max_{a_i \in A} \{\min_{b_j \in B} \{d(a_i, b_j)\}\}, \max_{b_j \in B} \{\min_{a_i \in A} \{d(b_j, a_i)\}\})$, one of the most well known distances for sets, is a metric if $d(a, b)$ is a metric.

The more complicated option is to consider a set of relations between elements of A and B , $R = \{R_i | R_i \subseteq A \times B\}$ and compute the distance between A and B based on the relation $R_i \in R$ that minimizes a distance on the elements of R_i . Here the relations may be surjections, fair surjections, linkings, or matchings (Kalousis 2007). Of these, the proposal by Ramon and Bruynooghe (2001) is a metric: as R it considers matchings in which each element of the two sets is associated with at most one element of the other set. The distance, defined as $d(A, B) = \min_{R_i \in R} (\sum_{(a_i, b_j) \in R_i} d(a_i, b_j) + (|B - R_i(A)| + |A - R_i^{-1}(B)|) \frac{M}{2})$ adds a $M/2$ penalty for the elements of A and B that do not participate in the relation R_i , where M is the maximum possible distance between two elements.

For the `Sequence()` type constructor, the edit-distance approach can be used. While this has typically been used for sequences of alphabet symbols, i.e., `Sequence(Discrete(Alphabet))`, Kalousis et al. (2007) have recently suggested an extension towards sequences of arbitrary complex objects on which a distance d is defined. Given two sequences $A = [a_1 \dots, a_m]$ and $B = [b_1 \dots, b_n]$, an alignment of A and B is a pair of sequences A' and B' of equal length $l \geq \max(n, m)$, constructed from the initial sequences by insertion of gaps, $-$. In an alignment, an element from A/B can be aligned to a gap (insert/delete operation) or two elements from A resp. B can be aligned with each other (replace operation).

The cost of an alignment is simply the sum of the cost of all operations used to derive the alignment, where the cost of the replace operation is $c(x, y) = d(x, y)$

and the cost of the insert and delete operations is a constant, i.e., $c(x, -) = c(-, y) = \alpha$, called the gap penalty. The alignment-based edit distance, $d_E(A, B)$, of two sequences A and B is then simply the minimum cost overall possible alignments of the two sequences, i.e., the cost of the lowest cost sequence of operations that turns the first sequence into the second. The edit distance can be computed using dynamic programming (Durbin et al., 1998) in time $O(mn)$.

Generic Distance-Based Algorithms. It is quite easy to formulate generic distance-based algorithms for data mining, which have the distance as a parameter. For example, hierarchical agglomerative clustering only makes use of the distances between the objects clustered and distances between sets of such objects. The latter can be based on single, complete or average linkage.

Clustering algorithms that make use of cluster prototypes, such as the k -means/medoids algorithm, require in addition a prototype function to be defined on the type of objects clustered. A closed-form prototype is desirable since its computation is more efficient and supports the version of the k -means algorithm parameterized with the distance and prototype functions. If no closed-form is known, the prototype of a cluster can be computed as the element of the cluster that has the lowest average (squared) distance to the other objects in the cluster, and the distance-parameterized version of (k -means like) k -medoids applies.

Distance-based prediction algorithms are also easy to formulate in a generic way. For a predictive problem of type $T_i \rightarrow T_j$, the nearest neighbor method applies as long as we have a distance on T_i . To make a prediction for a new instance, the distance between the (descriptive part of the) new instance and the training instances is calculated. The target part is copied from the nearest training instance and returned as a prediction. To use the k -nearest neighbor algorithm, we also need a prototype function on the target data type: the prediction returned is the prototype of the target parts of the k nearest (in the description space) instances.

9.2 Kernels and Kernel Methods

Kernels for a complex data type T can be derived from kernels for the simpler data types used to form T , analogously to distances. The manner in which the kernels for the simpler types are combined depends on the type constructor used to form T . As for distances, the easiest case is the one for tuples. A kernel function k on tuples of type $\text{Tuple}(T_1, \dots, T_n)$ can be composed from kernel functions k_i on types T_i by adding up the kernels for each tuple component $k(x, y) = k((x_1, \dots, x_n), (y_1, \dots, y_n)) = \sum_{i=1}^n k_i(x_i, y_i)$.

One possibility for kernels on sets is the product kernel, defined as $k(A, B) = \sum_{x \in A, y \in B} k(x, y)$. This is a special case of the convolution kernel proposed by Haussler (1999), which defines kernels of composite objects based on a relation between an object and its parts. Sets and multi-sets are also a special case of abstractions, which are mappings to the set of (non-negative) reals: sets are mappings to $\{0, 1\}$ ($A(x) = 1$ if $x \in A$), while multi-sets are mappings to

non-negative integers ($B(x) = 2$ if x occurs in B twice). Kernels for abstractions can be defined by $k(A, B) = \sum_{x \in A, y \in B} A(x)B(y)k(x, y)$ (Lloyd 2003).

Analogously to the manner in which edit-distances on strings have been adapted to work on sequences on structured objects (Kalousis et al. 2006), kernels on strings have been adapted to work on sequences on structured objects (Woznica et al. 2006). In particular, the “Contiguous Sublist” and the “Longest Common Sublist” kernels have been adapted to structured objects. A more detailed treatment of kernels on structured data is given by Gaertner (2003).

Kernel methods are also generic in the sense that any kernel can be used with any kernel algorithm, thus the kernel function can be seen as a parameter to the algorithm. We will not discuss here in detail any of the kernel methods, except to mention the most commonly used methods: support vector machines (SVMs), ridge regression and spectral clustering. As we have seen, kernels for structured data can be derived in a principled manner. In addition, special kernels have been designed for text, images, sequences and graphs, thus making kernel methods applicable to many kinds of data.

9.3 Features and Feature-Based Methods

The vast majority of data mining methods operate on a feature-based representation of data. These include, among others, decision trees and rules, linear equations and discriminants, and probabilistic methods (naive Bayes, Bayes nets). We will not discuss these here, but all of them assume that data reside in a single table, with columns representing features and rows representing data points (also referred to as instances and examples). Methods from machine learning, pattern recognition and statistics alike make use of this representation.

In this (sub)section, we briefly discuss how to derive features for structured objects in a principled manner. So far, this issue has been considered in more detail by subcommunities dealing with the analysis of more specific forms of data, such as image processing. The notable exception has been the relational data mining (Džeroski and Lavrač 2001) community, where propositionalization, introduced by Lavrač and Džeroski (1994), explicitly deals with the construction of features from (possibly structured) data represented relationally.

A large body of work exists in the machine learning and data mining communities that goes under the heading of feature construction and feature extraction (Liu and Motoda 1998), jointly referred to as feature discovery or feature transformation. Feature transformation is defined as the process in which a new set of features is constructed. If the new features are constructed by logical operations (e.g., conjunction or disjunction) on the original features, we speak about feature construction. If functional mappings are used instead (e.g., linear combinations), we speak about feature extraction. The crucial assumption is that the set of objects is originally already represented by a set of features, which are further combined to obtain new features. In our discussion, we do not make this assumption; rather the objects considered are of an arbitrary data type.

To construct the features for a type T , we proceed as follows. If the type T is primitive (Boolean, Discrete(S) or Real), a single feature of that type is

generated; we write $Features(T) = \{UniqueID : T\}$, where the feature of type T is assigned a unique identifier $UniqueID$. For a type $Tuple(T_1, \dots, T_n)$, the set of features generated is the union of the set of features derived for each of the component types, i.e., $Features(T) = \cup_{i=1}^n Features(T_i)$.

To describe a set of objects $t = \{t_1, \dots, t_k\}$ of type T , the most general approach would be to view t as a sample from a probability distribution p_T and specify the (joint) probability distribution as approximated from that sample. To describe t in terms of features, we need to consider features for describing probability distributions. If we assume the objects of type T can be described by a set/vector of features over the space $Features(T)$, we need to specify a probability distribution over that space. A simplified approach to this problem is to specify the marginals of this distribution, i.e., the distributions of the values for each feature.

For features of type $Discrete(S)$, where $S = \{s_1, \dots, s_m\}$, m features suffice to describe the distribution completely, namely $P(s_i)$, $i = 1, \dots, m$. Oftentimes, the mode of the distribution would be included as a feature, which is also of type $Discrete(S)$. The cardinality of the set $|t|$ is also a feature to be included. For real-valued features, their minimum and maximum values can be used as features, as well as the mean, standard deviation, and median: these are known as aggregates in relational databases. Histograms may be used if a more accurate description of the distribution is necessary.

For strings (sequences of objects of type $Discrete(S)$), n -grams are often used as features. These count the number of occurrences of each of the letters in the alphabet (1-grams), pairs of letters (2-grams), triplets of letters (3-grams), and so on. For time series (sequences of objects of type $Real$), many different features can also be constructed by using techniques from signal processing, such as the Fourier transform (Bracewell 1965) to the frequency domain or by wavelet analysis (Mallat 1999).

For objects of type $Sequence(T)$, assume we can represent objects of type T with $Features(T)$. We hence consider feature construction on an object of the type $Sequence(Tuple(F_1, \dots, F_k))$, where F_i are the types of the features from $Features(T)$. In analogy to the approach taken for sets, where the marginals were used to represent the probability distribution, we replace the type $Sequence(Tuple(F_1, \dots, F_k))$ with $Tuple(Sequence(F_1), \dots, Sequence(F_k))$ and derive features from this, resulting in the feature set $Features(Sequence(T)) = \cup_{i=1}^n Features(Sequence(F_i))$.

We have already mentioned earlier that features may be derived through the use of domain or background knowledge. Background knowledge can be thought of as a set of mappings. Assuming we are given a data type T for the data we are analyzing, as well as some additional data types T_i , as well as background knowledge consisting of a set of mappings $b_i : T \rightarrow T_i$.

In addition to the features $Features(T)$ that can be derived from T directly as discussed earlier in this section, features can be generated also from each of the types T_i . Suppose each of T_i gives rise to $Features_i(T_i)$. For an object t

of type T , besides $Features(t)$, the feature-based representation would include $Features_i(b_i(t))$.

For simplicity we have assumed one level of background knowledge only: all mappings apply to type T and yield an object of a type other than T . We can easily imagine that the background knowledge mappings apply to, as well as, yield objects of different types. The mappings would then be composed according to their type signatures. In that case, objects such as $b_i(b_j(t))$ and features thereof will have to be considered.

Let us conclude by noting that a huge number of features may be generated in this fashion. Hence we expect such feature generation to be used in conjunction with feature ranking and selection, where pattern discovery can be viewed as a form of the latter. For example, instead of using all possible n -grams as features to describe a sequence, only those that occur with a frequency exceeding a certain threshold may be used. There is a strong link between feature construction and (frequent) pattern discovery: the latter has been used to generate features for predictive modelling both in the context of frequent itemsets (Cheng et al. 2007) and frequent Datalog queries (King et al. 2001).

9.4 Refinement Orders and (Frequent) Pattern Discovery

In the most general formulation of pattern discovery, we can have an arbitrary language/class of patterns, a matching/covering relationship which corresponds to the interpreter for the given class of patterns, and an arbitrary evaluation function to optimize. In practice, the most commonly encountered pattern discovery task is the task of mining frequent patterns, where the evaluation function is frequency. For this task, the language of patterns is commonly the same (or very close) to the language of the data (type) considered.

When we mine frequent itemsets, the patterns - itemsets - are expressed in exactly the same language as the data - transactions. The same holds when mining frequent (sub)strings, (sub)sequences of structured terms or (sub)graphs. Consequently, the matching relationship used is a syntactic subsumption relationship, defined on the data type in question. For frequent itemsets this is the subset relation (on transactions), for the other data types this is analogously the substring, subsequence, and subgraph relationship.

The prototypical algorithm for mining frequent patterns starts its search with the empty pattern (set/sequence/graph), which is always frequent. It then proceeds levelwise, considering at each level the refinements of the patterns from the previous level and testing their frequencies. Only frequent patterns are kept for further refinement: Due to the anti-monotonicity of frequency, no refinement of an infrequent pattern can be frequent.

The key to a generic algorithm for mining frequent patterns that would work for arbitrary data types is to derive a subsumption/refinement relation from the definition of the type and subsumption relations for the component types, in much the same fashion we discussed above for distances, kernels and feature sets. For the primitive data types, subsumption is defined as follows. For the type $Discrete(S)$, a pattern is a subset of S and the \leq_{ref} relation corresponds to the \subseteq

relation. The data type Boolean can be viewed as a special case of Discrete(S), with $S = \{false, true\}$ ($\{0, 1\}$). For the type Real, a pattern is an interval of the form (a, b) : we say $(a, b) \leq_{ref} (c, d)$ iff $a \geq c$ and $b \leq d$, i.e., $(a, b) \subseteq (c, d)$ if we view the intervals as sets of real numbers.

For two tuples x and y of type $\text{Tuple}(T_1, \dots, T_n)$, such that $x = (x_1, \dots, x_n)$ and $y = (y_1, \dots, y_n)$, we have $x \leq_T y = \bigwedge_{i=1}^n (x_i \leq_{T_i} y_i)$. For two sets X and Y of objects of type T , we have $X \leq_{ref} Y$ iff there is a subset Y' of Y such that each element $x_i \in X$ can be paired with an element $y_i \in Y'$ such that $x_i \leq_T y_i$. For two sequences X and Y of objects of type T , we have $X \leq_{ref} Y$ iff there is a subsequence Y' of Y of the same length as X , such that $(x_i \leq_T y'_i)$ for all i .

10 Towards a Language for Data Mining and Knowledge Discovery

Based on the elements presented above, we outline here a vision of a language for data mining and knowledge discovery. First-order citizens of the language would include data types, data sets, and generalizations (patterns, models, clusterings, probability distributions), as well as data mining algorithm components, such as distance/cost functions, feature sets, kernels and refinement operators.

We envisage an interpreted language of declarative nature, incorporating some features from both functional programming (Thompson 1999) and logic programming (Lloyd 1987). For example, storing and querying structured data, database style, would be nicely supported by the logic programming side of the language. The same would hold for materialized collections of patterns/models, when queries concern the data part thereof.

On the other hand, the functional programming side would support the manipulation of the function aspects of patterns/models, such as the retrieval of the function aspect of a pattern/model from the data aspect or deriving predictions through the application of a given predictive model to new data. It would support operations on patterns and models, such as creating a mixture model from a given set of probability distributions. It would also support the derivation of the basic data mining algorithm ingredients for more complex data types from those of the component types. Finally, it would support the composition of different data mining operations into knowledge discovery scenarios.

10.1 Data and Background Knowledge

The data part of the language would be close in spirit to deductive databases (Lloyd 1983). A database in this setting would contain a set of data types T_1, \dots, T_n , as well as (some) datasets D_{ij} of objects of (each of) these types T_i . Background knowledge, consisting of a set of mappings $b : T_i \rightarrow T_j$, can also be included.

Each dataset can be viewed as a predicate. Each background knowledge mapping can be also viewed as a predicate $b(T_i, T_j)$. Assuming that the data are

represented relationally (in a flattened form), queries on data would be very similar to Datalog queries.

In our setting, data types are represented explicitly and can also be the object of manipulation. Recall that types are constructed from primitive data types (Boolean, Discrete(S), and Real) using type constructors (Tuple, Set, Sequence). In this context, it is important to note the relationships between different data types, which can lead to a taxonomy/ontology of data types.

For example, we can have a data type molecule, which is a special case of labeled graphs. Molecular fragments, which can also be defined as a type, are substructures (linear paths) in such graphs. Explicitly representing and reasoning with such information can be very useful, for example, to determine the applicability of different data mining algorithms to different data types: an algorithm for mining frequent subgraphs can be used for finding frequent molecular fragments.

10.2 Generalizations

Patterns, models, clusterings and probability distributions (which we collectively refer to as generalizations) are first-class citizens in this framework. Recall that they all have both a data and a function part. Classes of generalizations can be defined, for which an interpreter can map the data part of a generalization to its function part.

The function part of a generalization is typed according to the underlying data type(s), for example, $\text{Sequence}(\{A, C, T, G\}) \rightarrow \text{Boolean}$. For the data part, a type definition needs to be given, so that the generalizations can be explicitly stored and queried as data objects. The symbols that can appear in the type definition depend on the underlying data type(s), as well as the class of generalizations. If we consider classification trees that classify tuples of Boolean features $X = (x_1, \dots, x_n)$ into a Boolean class, internal nodes of the trees would be labeled with a feature name $x_i \in X$ while leaves will be labeled with one of $\{\text{true}, \text{false}\}$.

We can imagine sets of decision trees stored in a ‘dataset’ of the appropriate type. For example, the set of trees in a random forest can be stored in such a dataset. The trees need not all be generated by machine: we can give students, as part of their machine learning exam, the task to generate a decision tree for a certain (small) set of examples. All of the answers we get from the class of students would form such a dataset of decision trees as well.

Datasets of generalizations of the kind outlined above can be queried in much the same fashion as ordinary datasets. For example, datasets of decision trees can be queried much like datasets of ordinary trees. We might ask, e.g., for all the trees that contain a certain feature at the root node.

The set of solutions to a data mining query, i.e., the set of generalizations of a certain class satisfying a given set of constraints would also constitute a dataset of generalizations. For example, all the decision trees of size at most 10 leaves with classification error of at most 10% would form a dataset of trees. Another example would be the set of all itemsets with frequency higher than 10% and

cost lower than 100\$: Note that the latter also refers to a cost function on the data part of the pattern. Additional queries, called post-processing queries, can be posed after the data mining results have become part of a dataset.

10.3 Cross-over Queries

Cross-over queries apply generalizations to a given datum or a dataset. If the source data type is T_d and the generalization maps to type T_c , we can think of cross-over queries as producing pairs of type (T_d, T_c) . Recall that patterns map to Booleans, clusterings to integers in the range $1, \dots, k$ (where k is the number of clusters), probability distributions/densities to non-negative Reals, and predictive models to an arbitrary data type T_c . Since cross-over queries most often involve predictive models, we refer to the application as a prediction join.

More specifically, the application of a generalization to a given datum refers to the function part of the generalization. If the function part is explicitly stored (e.g., as a stored procedure), it would be called directly. Otherwise we would make use of the interpreter for the particular class of generalizations.

Once we have the results of applying a generalization to a dataset, the value of an evaluation function can be computed for that generalization. For example, for a classification model, the classification error can be computed. For a clustering, the intra-cluster variance can be calculated.

10.4 Generic Data Mining: Components and Algorithms

We also propose that the basic components of data mining algorithms, i.e., distance and prototype functions, kernel functions, feature sets, and refinement operators, are also first-class citizens of our language. Suppose we have a set of distance functions, e.g., a set of weighted Euclidean distances over tuples of reals, each represented by a tuple of weights. We can query this set and look for distances that give a very high weight (over a threshold) to a selected feature.

Generic data mining algorithms, such as the ones for distance-based prediction discussed in the previous section, could then be used, parameterized with the selected basic components. We envisage several generic algorithms for each of the major approaches (distance-based, kernel, and feature based). For example, distance-based methods would include hierarchical clustering (agglomerative and divisive), k -means/medoids clustering, and k -nearest neighbor prediction.

Generic feature-based algorithms for predictive data mining would be parameterized with feature sets on the descriptive side and distance functions for the target side. If the prediction problem is predicting objects of type T_c from objects of type T_d , the algorithm will need a set of features $Features(T_c)$ and a distance on T_d as input. Algorithms for learning predictive clustering trees (Blockeel et al. 1998) and rules (Ženko et al. 2006) have come closest to this paradigm.

We expect default basic components to be associated with each data type and/or generic algorithm. For example, a default distance can be associated with each data type. For the primitive data types, these have been discussed

earlier (Section 7.2). For compound data types, if not specified explicitly, default distances can be derived using the methodology outlined in Section 9.1.

10.5 Constraints and Data Mining Queries

Once we have defined a data mining task and selected the associated basic components, we can define precisely the evaluation function(s) to be used. For example, we can only define predictive error once we have defined a distance/cost function on the target data type of a predictive modelling task. Different evaluation functions can be defined by selecting different basic components (distances). For probabilistic classification, one can optimize squared loss or log loss, depending on the underlying distance on probability distributions over discrete variables. Note that the above are independent of the class of predictive models used.

Besides evaluation constraints, language constraints, which can be subsumption-based or based on cost functions, can also be used. These are defined for a specific class of generalizations. Subsumption constraints involve the generality/refinement order on the data part of the generalization. For example, a specific item has to appear in the discovered frequent itemsets or the classification tree has to include a given subtree at the root. Examples of cost functions for language constraints include the size or depth of a decision tree and the cost of an itemset. Note that we can also have evaluation constraints that are specific for a given class of generalizations. For instance, we can require each leaf of a decision tree to cover at least 10 examples, have accuracy of its majority class higher than a threshold, or have a majority class frequency higher by a given margin than the frequency of any other class (Nijssen and Fromont 2007).

Similar to the ontology of data types mentioned above, we can imagine having a hierarchy/an ontology on generalizations. Evaluation and language cost functions can be defined at different levels in this ontology. The most general ones will apply, say, for all models, while more specific ones will apply for a given class of models only, e.g., for decision trees.

When a user defines a data mining task, she will then have available a choice of primitives (evaluation functions, language cost functions, and subsumption relations) appropriate to the task at hand. The primitives can be used to form individual constraints, which can then be combined to form inductive (data mining) queries. Recall that we have Boolean (hard) constraints, optimization constraints and soft constraints. Boolean constraints can be combined via logical operations to form complex queries. Note that the use of multiple evaluation/language cost functions entails a multi-objective optimization problem, to which different approaches can be applied as outlined in Section 8.4.

The design of data mining algorithms to solve arbitrary inductive queries composed along the lines described above is still much of an open issue. So far, most of the algorithms for constraint-based data mining have focussed on mining frequent patterns (in structured data) under frequency constraints (Boulicaut et al. 2005). Mining closed patterns and mining under (language) cost constraints has also been considered in this setting. Since all of the above are anti-monotonic,

combinations thereof are also anti-monotonic and the generic algorithm for mining frequent patterns (discussed in Section 9.4) applies.

Interest in the constraint-based mining of predictive models has increased recently. A number of methods have been proposed that take into account accuracy and size constraints in decision trees (Garofalakis et al. 2003), as well as subsumption constraints (Struyf and Džeroski 2006). Constraints (including subsumption and accuracy) have also been considered in the context of learning polynomial equations (Džeroski et al. 2005). However, unlike ‘complete’ frequent pattern mining approaches, which return the complete/optimal set of solutions, these approaches are heuristic and give no guarantees. Only recently have ‘complete’ approaches been considered for predictive models, e.g., for learning optimal decision trees (Nijssen and Fromont 2007).

10.6 Re-using the Results of Learning

Recall that the set of solutions to an inductive query produced by a data mining algorithm constitutes a dataset of generalizations (e.g., patterns/models) and can be stored as such. It then becomes available for further queries, be it post-processing queries, cross-over queries or inductive queries. Actually, post-processing queries can be viewed as just data queries on the data part of the generalizations.

One way to use cross-over queries is to apply learned models (patterns) on a new set of data to produce a new (feature-based) representation. For example, as discussed in Section 9.3 frequent patterns (discovered by data mining) have already been used as features for predictive modelling. This has been done both for itemsets/propositional representations (Cheng et al. 2007) and for structured objects/relational representations (King et al. 2001).

Models can also be used to generate new features, either directly or indirectly. If the target type of a model is a primitive type or a vector thereof, the model’s predictions can be used directly as features. If it is structured, feature construction thereon can be applied as outlined in Section 9.3. The above can be done for any model, irrespective of its representation.

For some classes of models, features can be generated from the structure of the data part of the model. For instance, if we have a decision tree, we can take fragments thereof, such as paths which represent a conjunction of conditions, and make these new features. A similar approach has been used by Srinivasan and King (1999) to extract features from a logic program, then use these for predicting biological activity of molecules.

We can also pose inductive queries on sets of patterns/models. We take as input the data part of the patterns/models. It is important to note that the ability to mine structured data is of crucial importance here. Namely, patterns and models are often structured objects, even if the data they were generated from is flat/propositional: Take for example decision trees and Bayesian networks derived from a propositional dataset.

Termier et al. (2006) first learn gene networks from microarray data through a process that generates many different networks. They then apply frequent

pattern mining to extract DAGs (directed acyclic graphs) which commonly occur in the networks. A human expert then only needs to look at and assess the biological meaning of these smaller components of the gene networks.

The above discussion addresses directly issues raised by Siebes (2006) as priority issues to be solved in order to achieve truly inductive database functionality. The paragraphs immediately above refer to what Siebes calls ‘Models on Models’ or ‘Mining on Models and Patterns’. The earlier part of the discussion refers to what Siebes calls ‘Models for Models’. Both are related to meta-learning (Vilalta and Drissi 2002), where the results of (base-level) learning are used as input to a further learning process.

More complicated ways are possible of re-using learned patterns and models. In some ways, these would depart from the basic data mining tasks that we have addressed earlier in this article. For example, a given set of data can be used to improve an existing predictive model: This task is referred to as theory revision. Here we want to find an improved model, which however is not discovered from scratch but is based on the given model. One possibility is to formulate this in terms of subsumption constraints. Džeroski et al. (2005) discuss a scenario in which a previously discovered polynomial is used in a sub-polynomial constraint in the discovery of more complex polynomials. In general, we want the revised model to be similar to the original model, and constraints based on (syntactic or semantic) similarity of patterns/models might prove to be useful.

10.7 Operations on Generalizations

It would be desirable to support some operations on generalizations in a data mining language. An example would be the operation that combines a set of regression models by taking the average prediction of individual models. A set of classification models would be combined via majority voting and a set of probabilistic classification models by probability distribution voting/summation. Another example would be the weighted summation of a number of probability distributions to obtain a mixture model. Finally, clustering aggregation (Gionis et al. 2005) takes as input a set of clusterings and finds a clustering that agrees as much as possible with the given clusterings.

The first two examples above are easy to implement, especially in functional programming languages. We can operate on the functional part of the generalizations directly, and obtain, e.g., $r(x) = (r_1(x) + \dots + r_k(x))/k$ in the first example, where $r_i(x)$ are the individual regression models (the function parts thereof). For certain classes of generalizations, these operations can be done on the data part thereof in closed form: if $r_i(x)$ were linear models represented by the coefficients for each of the features x_j used, we could simply add these up.

Defining operations on generalizations with a clear semantics goes in the direction of defining an algebra for data mining, analogous to relational algebra. Mannila (2001) outlined a proposal of an algebra for probabilistic (mixture) models, with the basic operations of projection, selection, union and join analogues to the operations in the relational model. A probabilistic model is viewed

as a relation: selection corresponds to partial evaluation of the model, projection to marginalization over the variables projected away, union corresponds to mixing, and applying a model to an ordinary relation to (prediction) join.

10.8 Integration Aspects, Compositionality, and Scenarios

So, how would a user go about solving a specific data mining problem in the framework outlined here? She would start with the data: Define data type(s), or choose from pre-defined ones, then load a dataset. Along with the data type(s), distances/prototypes, features, kernels and refinements may be defined (either directly by the user or in a semi-automated fashion from the definitions of the data types).

The data mining task has to be specified next: This can be one of pattern discovery, estimating a probability distribution, clustering or prediction. A different type of generalization (a set of patterns, a probability distribution, a clustering or a predictive model) is produced as output, depending on the task. These are parameterized in the first instance with the types of data they operate on.

Once a data mining task and the underlying data types are selected, a class of generalizations has to be selected. We expect that some generic classes (e.g., trees) will be predefined, parameterized with the appropriate basic components (e.g., feature sets). Evaluation/language cost functions and constraints have to be selected next, based on distance/cost functions and refinement orders on the underlying data/generalization types: These constitute an inductive query.

Integration. The language we envision would support the creation of data mining algorithms capable of solving a variety of inductive queries on a variety of data types involving a variety of evaluation and language constraints. The key to this would be the support for different data types, the explicit treatment of generalizations as both data objects and functions, the explicit representation and manipulation of basic components of data mining algorithms, as well as constraint primitives (evaluation and language cost functions). Data mining algorithms implemented in this environment would be tightly integrated into an overall knowledge discovery language, where outputs of one data mining operation can be used as input to another.

Loose integration of externally developed data mining algorithms would also be possible. For an algorithm to be plugged in, we would need a precise specification of the data mining task addressed, the underlying data type(s), and the class of generalizations considered. If the algorithm is constraint-based, the evaluation and language primitives used, and the types of constraints supported should also be specified. For example, MolFea (Kramer and De Raedt 2001) addresses the task of finding frequent patterns in the form of linear molecular fragments in data that consists of molecular structures in the form of labeled graphs and takes into account subsumption language constraints and frequency evaluation constraints.

Compositionality. Compositionality is the technique of constructing complex analyses by using a collection of standard operations as building blocks, with the

outputs of some operations serving as inputs to other operations (Ramakrishnan et al. 2005). Relational algebra, for example, includes the operations of selection, projection, join, union and difference that take as input tables and produce tables as output. In the context of data mining, it has so far been a largely open issue what are the operators of interest, their inputs and outputs.

We argue that data mining operators corresponding to the data mining tasks we have defined and discussed here should definitely be included in an algebra for data mining. Whether or not (and which) operations of lower granularity should be included can be debated on further and depends on whether we want to focus on knowledge discovery or data mining. In a knowledge discovery language a coarser granularity may be preferred, while for data mining a finer granularity might be necessary.

We have described data mining tasks/operations, as well as the basic ingredients of data mining algorithms, in terms of their signatures, i.e., the domain and ranges of the functions they are computing. Not all operations can be meaningfully combined in all possible ways and signatures provide us with some guidance on what combinations are meaningful. For example, the output of a frequent pattern mining algorithm for data type T applied to a dataset D_1 of objects of type T can be used as the input to a cross-over operation (together with another dataset D_2 of objects of type T).

The signatures of data mining operators can be organized in a hierarchy (Ramakrishnan et al. 2005). At the higher level, the signatures are described in general terms, such as pattern or model. In the lower levels they may be specialized for certain types/classes of patterns or models. Following the same approach makes sense in our case as well. We propose to lift the hierarchy described there, which works only for propositional data, to the case of mining structured data. The ontology of structured data types would be taken into account as well.

Scenarios. Real-life applications of data mining typically require interactive sessions and involve the formulation of a complex sequence of inter-related inductive queries (including data mining operations), which we will call a KDD scenario (Boulıcaut et al. 1999). Some of the inductive queries would generate or manipulate patterns, others would apply these patterns to a given dataset to form a new dataset, still others would use the new dataset to build a predictive model. The ability to formulate and execute such sequences of queries crucially depends on compositionality, the ability to use the output of one query as the input to another.

KDD scenarios can be described at different levels of detail and precision and can serve multiple purposes. At the lowest level of detail, the specific data mining algorithms used and their exact parameter settings employed would be included, as well as the specific data analyzed. Moving towards higher levels of abstraction, details can be gradually omitted, e.g., first the parameter setting of the algorithm, then the actual algorithm may be omitted but the class of generalizations produced by it can be kept, and finally the class of generalizations can be left out (but the data mining task kept). On the data side, one might move from the specification of an actual dataset to a specification of the underlying

data type and further to data types that are higher in the hierarchy/ontology of data types. Having ontologies of data types, data mining tasks, generalizations and data mining algorithms would greatly facilitate the description of scenarios at higher abstraction levels: the abstraction can proceed along each of the respective ontologies.

At the most detailed level of description, KDD scenarios can serve to document the exact sequence of data mining operations undertaken by a human analyst on a specific task. This would facilitate, for example, the repetition of the entire sequence of analyses after an erroneous data entry has been corrected in the source data. At higher levels of abstraction, the scenarios would enable the re-use of already performed analyses, e.g., on a new dataset of the same type. We thus argue that the explicit storage and manipulation of scenarios (e.g., by reducing/increasing the level of detail) would greatly facilitate the KDD process as a whole, reduce human effort and thus alleviate a major bottleneck in applying KDD in practice.

11 Related Work

When attempting to formulate a general framework for data mining, the potential set of related work items is dangerously large. Here we will give a biased sample of what we consider related work. Parts of it have been mentioned previously, while others have not been explicitly mentioned above even though they have made an intellectual influence during the writing of this article.

Let us start with inductive databases and constraint-based data mining. Since the notion of inductive databases was introduced, a significant body of research has grown on these two topics: A survey can be found in the book edited by Boulicaut et al. (2005). An earlier collection of papers focussing on constraint-based data mining was edited by Bayardo (2002).

Data mining query languages are also directly relevant: A survey article is presented by Boulicaut and Masson (2005). A more recent proposal for an SQL-based data mining query languages, which allows for the integration of various data mining operations at the data level, has been given by Kramer et al. (2006). Finally, the IQL language proposed by Nijssen and De Raedt (2007/*this volume*), is very close in spirit to the discussion presented here: it recognizes the importance of functions and extends tuple relational calculus with a function and a typing system. However, it only allows for loose integration of data mining algorithms and does not support the creation of new algorithms.

Another way to recognize the importance of functions is to use higher-order logic or functional programming to facilitate the implementation of data mining algorithms (for mining structured data). Lloyd (2003) uses higher-order logic to define structured data types and principled ways of constructing distances, features (which he calls predicates) and kernels. Allison (2004) uses functional programming to define data types and type classes for models (where models include probability distributions, mixture models and decision trees) that allow for models to be manipulated in a precise and flexible way.

Formulating an algebra for data mining that would be the equivalent of Codd's relational algebra for databases is probably the most ambitious goal in the context of the discussion presented here. The 3W-model (Johnson et al. 2000) was among the first to take an algebraic view on data mining: A refined version has been presented recently by Calders et al. (2006b). Mannila (2001) presented an algebraic view on operations with mixture models. Siebes (2006) discusses how to lift relational algebra to patterns and models. Finally, the compositionality of data mining operators, as discussed by Ramakrishnan et al. (2005), can be expected to play a crucial role in the general framework.

Acknowledgments. This work was supported by the IQ project (IST-FET FP6-516169). Thanks are due to the members of the project for providing the intellectual background for this work, as well as numerous discussions on related issues. Special thanks to Jan Struyf, who helped with this article in various ways, which among other things included a detailed proofreading and search for references. Thanks also to Alexandros Kalousis and Ljupčo Todorovski for reading this text at short notice, as well as Marko Bohanec and Bernard Ženko for commenting on portions thereof. Thanks as well to Martin Erwig and John Lloyd for the useful discussions on functional programming languages and their use for data mining.

References

1. Agrawal, R., Imielinski, T., Swami, A.: Mining association rules between sets of items in large databases. In: Proc. of the ACM SIGMOD Conf. on Management of Data, pp. 207–216. ACM Press, New York (1993)
2. Aho, A.V., Ullman, J.D., Hopcroft, J.E.: Data Structures and Algorithms. Addison-Wesley, Reading, MA (1983)
3. Allison, L.: Models for machine learning and data mining in functional programming. *Journal of Functional Programming* 15(1), 15–32 (2004)
4. R. Bayardo (ed.) Constraints in data mining. Special issue of SIGKDD Explorations, 4(1) (2002)
5. Bishop, C.M.: Pattern Recognition and Machine Learning. Springer, Berlin (2006)
6. Bistarelli, S., Bonch, F.: Interestingness is not a Dichotomy: Introducing Softness in Constrained Pattern Mining. In: Jorge, A.M., Torgo, L., Brazdil, P.B., Camacho, R., Gama, J. (eds.) PKDD 2005. LNCS (LNAI), vol. 3721, Springer, Heidelberg (2005)
7. Blockeel, H., De Raedt, L., Ramon, J.: Top-down induction of clustering trees. In: Proc. of the 15th Intl. Conf. on Machine Learning, pp. 55–63. Morgan Kaufmann, San Mateo, CA (1998)
8. Boulicaut, J.-F., Jeudy, B.: Constraint-based data mining. In: Maimon, O., Rokach, L. (eds.) *The Data Mining and Knowledge Discovery Handbook*, pp. 399–416. Springer, Berlin (2005)
9. Boulicaut, J.-F., Masson, C.: Data mining query languages. In: Maimon, O., Rokach, L. (eds.) *The Data Mining and Knowledge Discovery Handbook*, Springer, Berlin (2005)
10. Boulicaut, J.-F., Klemettinen, M., Mannila, H.: Modeling KDD processes within the inductive database framework. In: Mohania, M.K., Tjoa, A.M. (eds.) *DaWaK 1999*. LNCS, vol. 1676, pp. 293–302. Springer, Heidelberg (1999)

11. Boulicaut, J.-F., De Raedt, L., Mannila, H. (eds.): *Constraint-Based Mining and Inductive Databases*. Springer, Berlin (2005)
12. Bracewell, R.N.: *The Fourier Transform and Its Applications*. McGraw-Hill, New York (1965)
13. Calders, T., Rigotti, C., Boulicaut, J.-F.: A survey on condensed representations for frequent sets. In: Boulicaut, J.-F., De Raedt, L., Mannila, H. (eds.) *Constraint-Based Mining and Inductive Databases*, pp. 64–80. Springer, Berlin (2005)
14. Calders, T., Goethals, B., Prado, A.B.: Integrating pattern mining in relational databases. In: *Proc. of the 10th European Conf. on Principles and Practice of Knowledge Discovery in Databases*, pp. 454–461. Springer, Berlin (2006a)
15. Calders, T., Lakshmanan, L.V.S., Ng, R.T., Paredaens, J.: Expressive power of an algebra for data mining. *ACM Transactions on Database Systems* 31(4), 1169–1214 (2006b)
16. Cheng, H., Yan, X., Han, J., Hsu, C.-W.: Discriminative frequent pattern analysis for effective classification. In: *Proc. 23rd Intl. Conf. on Data Engineering*, pp. 716–725. IEEE Computer Society Press, Los Alamitos (2007)
17. Deb, K.: *Multi-Objective Optimization Using Evolutionary Algorithms*. Wiley & Sons, New York (2001)
18. De Raedt, L., Dehaspe, L.: Clausal discovery. *Machine Learning* 26, 99–146 (1997)
19. Dehaspe, L., Toivonen, H.: Discovery of frequent Datalog patterns. *Data Mining and Knowledge Discovery* 3(1), 7–36 (1999)
20. De Raedt, L.: A perspective on inductive databases. *SIGKDD Explorations* 4(2), 69–77 (2002a)
21. De Raedt, L.: Data mining as constraint logic programming. In: Kakas, A.C., Sadri, F. (eds.) *Computational Logic: Logic Programming and Beyond – Essays in Honour of Robert A. Kowalski, Part II*, pp. 113–125. Springer, Berlin (2002b)
22. Durbin, R., Eddy, S.R., Krogh, A., Mitchison, G.J.: *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, Cambridge (1998)
23. Džeroski, S.: Inductive logic programming in a nutshell. In: Getoor, L., Taskar, B. (eds.) *Statistical Relational Learning*, MIT Press, Cambridge, MA (2007)
24. Džeroski, S., Lavrač, N. (eds.): *Relational Data Mining*. Springer, Berlin (2001)
25. Džeroski, S., Todorovski, L., Ljubič, P.: Inductive queries on polynomial equations. In: Boulicaut, J.-F., De Raedt, L., Mannila, H. (eds.) *Constraint-Based Mining and Inductive Databases*, pp. 127–154. Springer, Berlin (2005)
26. Fayyad, U., Piatetsky-Shapiro, G., Uthurusamy, R.: Summary from the KDD-2003 panel – “Data Mining: The Next 10 Years”. *SIGKDD Explorations* 5(2), 191–196 (2003)
27. Friedman, J.H., Fisher, N.I.: Bump hunting in high-dimensional data. *Statistics and Computing* 9(2), 123–143 (1999)
28. Fayyad, U., Piatetsky-Shapiro, G., Smyth, P.: From data mining to knowledge discovery: An overview. In: Fayyad, U., Piatetsky-Shapiro, G., Smyth, P., Uthurusamy, R. (eds.) *Advances in Knowledge Discovery and Data Mining*, pp. 495–515. MIT Press, Cambridge, MA (1996)
29. Frawley, W.J., Piatetsky-Shapiro, G., Matheus, C.J.: Knowledge discovery in databases: An overview. In: *Knowledge Discovery in Databases*, pp. 1–30. AAAI/MIT Press, Cambridge
30. Gaertner, T.: A survey of kernels for structured data. *SIGKDD Explorations* 5(1), 49–58 (2003)
31. Garofalakis, M., Hyun, D., Rastogi, R., Shim, K.: Building decision trees with constraints. *Data Mining and Knowledge Discovery* 7(2), 187–214 (2003)

32. Getoor, L., Taskar, B. (eds.): Statistical Relational Learning. MIT Press, Cambridge, MA (2007)
33. Gionis, A., Mannila, H., Tsaparas, P.: Clustering aggregation. In: Proc. of the 21st Intl. Conf. on Data Engineering, pp. 341–352. IEEE Computer Society Press, Los Alamitos (2005)
34. Han, J., Kamber, M.: Data Mining: Concepts and Techniques. Morgan Kaufmann, San Francisco, CA (2001)
35. Hand, D.J., Mannila, H., Smyth, P.: Principles of Data Mining. MIT Press, Cambridge, MA (2001)
36. Haussler, D.: Convolution kernels on discrete structures. UC Santa Cruz, Technical Report UCS-CRL-99-10 (1999)
37. Imielinski, T., Mannila, H.: A database perspective on knowledge discovery. Communications of the ACM 39(11), 58–64 (1996)
38. Johnson, T., Lakshmanan, L.V., Ng, R.: The 3W model and algebra for unified data mining. In: Proc. of the Intl. Conf. on Very Large Data Bases, pp. 21–32. Morgan Kaufmann, San Francisco, CA (2000)
39. Kalousis, A., Woznica, A., Hilario, M.: A unifying framework for relational distance-based learning founded on relational algebra. Technical Report, Computer Science Department, University of Geneva (2006)
40. Kaufman, L., Rousseeuw, P.J.: Finding Groups in Data: An Introduction to Cluster Analysis. Wiley & Sons, New York (1990)
41. King, R.D., Karwath, A., Clare, A., Dehaspe, L.: The utility of different representations of protein sequence for predicting functional class. Bioinformatics 17(5), 445–454 (2001)
42. Kloesgen, W.: Data mining tasks and methods: Subgroup discovery: deviation analysis. In: Kloesgen, W., Zytkow, J.M. (eds.) Handbook of Data Mining and Knowledge Discovery, pp. 354–361. Oxford University Press, Oxford (2002)
43. Kramer, S., Aufschild, V., Hapfelmeier, A., Jarasch, A., Kessler, K., Reckow, S., Wicker, J., Richter, L.: Inductive Databases in the Relational Model: The Data as the Bridge. In: 4th Intl. Wshp. on Knowledge Discovery in Inductive Databases: Revised Selected and Invited Papers, pp. 124–138. Springer, Berlin (2006)
44. Lavrač, N., Kavšek, B., Flach, P.A., Todorovski, L.: Subgroup Discovery with CN2-SD. Journal of Machine Learning Research 5, 153–188 (2004)
45. Lavrač, N., Džeroski, S.: Inductive Logic Programming: Techniques and Applications. Ellis Horwood, Chichester (1994)
46. Liu, H., Motoda, H.: Feature Extraction, Construction and Selection: A Data Mining Perspective. Kluwer, Dordrecht (1998)
47. Lloyd, J.W.: Foundations of Logic Programming. Springer, Berlin (1987)
48. Lloyd, J.W.: An introduction to deductive database systems. Australian Computer Journal 15(2), 52–57 (1983)
49. Lloyd, J.W.: Logic for Learning. Springer, Berlin (2003)
50. Mallat, S.: A Wavelet Tour of Signal Processing. Academic Press, London (1999)
51. Inductive databases vision: Relational operations on models. Unpublished slides. In: Presented at the meeting of the cInQ project (December 2001)
52. Mannila, H., Toivonen, H.: Levelwise search and borders of theories in knowledge discovery. Data Mining and Knowledge Discovery 1(3), 241–258 (1997)
53. Michalski, R.S.: Knowledge acquisition through conceptual clustering: A theoretical framework and an algorithm for partitioning data into conjunctive concepts. Intl. Jnl. of Policy Analysis and Information Systems 4, 219–244 (1980)
54. Mitchell, T.M.: Generalization as search. Artif. Intell. 18(2), 203–226 (1982)

55. Nijssen, S., Fromont, E.: Mining optimal decision trees from itemset lattices. In: Proc. of The 13th ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining, ACM Press, New York (to appear, 2007)
56. Piatetsky-Shapiro, G., Djeraba, C., Getoor, L., Grossman, R., Feldman, R., Zaki, M.: What are the grand challenges for data mining? KDD-2006 Panel report. SIGKDD Explorations 8(2), 70–77 (2006)
57. Ramakrishnan, R., et al.: Data Mining: The Next Generation. In: Ramakrishnan, R., Agrawal, R., Freytag, J.-C. (eds.) Perspectives Wshp. – Data Mining: The Next Generation. Intl. Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany (2005)
58. Ramon, J., Bruynooghe, M.: A polynomial time computable metric between point sets. Acta Informatica 37(10), 765–780 (2001)
59. Shawe-Taylor, J., Cristianini, N.: Kernel Methods for Pattern Analysis. Cambridge University Press, Cambridge (2004)
60. Siebes, A.: Data mining in inductive databases. In: 4th Intl. Wshp. on Knowledge Discovery in Inductive Databases: Revised Selected and Invited Papers, pp. 1–23. Springer, Berlin (2006)
61. Srinivasan, A., King, R.D.: Feature construction with inductive logic programming: A study of quantitative predictions of biological activity aided by structural attributes. Knowledge Discovery and Data Mining 3(1), 37–57 (1999)
62. Struyf, J., Džeroski, S.: Constraint based induction of multi-objective regression trees. In: 4th Intl. Wshp. on Knowledge Discovery in Inductive Databases: Revised Selected and Invited Papers, pp. 222–233. Springer, Berlin (2006)
63. Termier, A., Tamada, Y., Imoto, S., Washio, T., Higuchi, T.: From closed tree mining towards closed DAG mining. In: Proc. of the Intl. Wshp. on Data Mining and Statistical Science, pp. 1–7 (2006)
64. Thompson, S.: Haskell: The Craft of Functional Programming. Add. Wesley (1999)
65. Tušar, T.: Design of an Algorithm for Multiobjective Optimization with Differential Evolution. M.Sc. Thesis. Faculty of Computer and Information Science, University of Ljubljana, Slovenia (2007)
66. Vilalta, R., Drissi, Y.: A perspective view and survey of meta-learning. Artificial Intelligence Review 18(2), 77–95 (2002)
67. Wagstaff, K., Cardie, C.: Clustering with instance-level constraints. In: Proc. 17th Intl. Conf. on Machine Learning, pp. 1103–1110. Morgan Kaufmann, San Francisco, CA (2000)
68. Woznica, A., Kalousis, A., Hilario, M.: Kernels on lists and sets over relational algebra: an application to classification of protein fingerprints. In: Proc. 10th Pacific-Asia Conf. on Knowledge Discovery and Data Mining, Springer, Berlin (2006)
69. Yang, Q., Wu, X.: 10 Challenging problems in data mining research. Intl. Jnl. of Information Technology & Decision Making 5(4), 597–604 (2006)
70. Ženko, B., Džeroski, S., Struyf, J.: Learning predictive clustering rules. In: 4th Intl. Wshp. on Knowledge Discovery in Inductive Databases: Revised Selected and Invited Papers, pp. 234–250. Springer, Berlin (2006)