Chapter 2
Representing Entities in the OntoDM Data Mining Ontology

Panče Panov, Larisa N. Soldatova, and Sašo Džeroski

Abstract  Motivated by the need for unification of the domain of data mining and the demand for formalized representation of outcomes of data mining investigations, we address the task of constructing an ontology of data mining. Our heavy-weight ontology, named OntoDM, is based on a recently proposed general framework for data mining. It represent entites such as data, data mining tasks and algorithms, and generalizations (resulting from the latter), and allows us to cover much of the diversity in data mining research, including recently developed approaches to mining structured data and constraint-based data mining. OntoDM is compliant to best practices in ontology engineering, and can consequently be linked to other domain ontologies: It thus represents a major step towards an ontology of data mining investigations.

2.1 Introduction

Traditionally, ontology has been defined as the philosophical study of what exists: the study of kinds of entities in reality, and the relationships that these entities bear to one another [41]. In recent years, the use of the term ontology has become prominent in the area of computer science research and the application of computer science methods in management of scientific and other kinds of information. In this sense, the term ontology has the meaning of a standardized terminological framework in terms of which the information is organized.
The ontological problem in general is focused on adopting a set of basic categories of objects, determining what (kinds of) entities fall within each of these categories of objects, and determining what relationships hold within and among different categories in the ontology. The ontological problem for computer science is identical to many of the problems in philosophical ontology: The success of constructing such an ontology is thus achievable by applying methods, insights and theories of philosophical ontology. Constructing an ontology then means designing a representational artifact that is intended to represent the universals and relations amongst universals that exist, either in a given domain of reality (e.g the domain of data mining research) or across such domains.

The engineering of ontologies is still a relatively new research field and some of the steps in ontology design remain manual and more of an art than craft. Recently, there has been significant progress in automatic ontology learning [31], applications of text mining [7], and ontology mapping [29]. However, the construction of a high quality ontology with the use of automatic and even semi-automatic techniques still requires manual definition of the key upper level entities of the domain of interest. Good practices in ontology development include following an upper level ontology as a template, the use of formally defined relations between the entities, and not allowing multiple inheritances [44].

In the domain of data mining and knowledge discovery, researchers have tried to construct ontologies describing data mining entities. These ontologies are developed to solve specific problems, primarily the task of automatic planning of data mining workflows [2, 24, 11, 22, 26]. Some of the developments are concerned with describing data mining services on the GRID [8, 5].

The currently proposed ontologies of data mining are not based on upper level categories nor do they have use a predefined set of relations based on an upper level ontology. Most of the semantic representations for data mining proposed so far are based on so called light-weight ontologies [33]. Light-weight ontologies are often shallow, and without rigid relations between the defined entities. However, they are relatively easy to develop by (semi)automatic methods and they still greatly facilitate several applications. The reason these ontologies are more frequently developed then heavy-weight ontologies is that the development of the latter is more difficult and time consuming. In contrast to many other domains, data mining requires elaborate inference over its entities, and hence requires rigid heavy-weight ontologies, in order to improve the Knowledge Discovery in Databases (KDD) process and provide support for the development of new data mining approaches and techniques.

While KDD and data mining have enjoyed great popularity and success in recent years, there is a distinct lack of a generally accepted framework that would cover and unify the data mining domain. The present lack of such a framework is perceived as an obstacle to the further development of the field. In [52], Yang and Wu collected the opinions of a number of outstanding data mining researchers about the most challenging problems in data mining research. Among the ten topics considered most important and worthy of further research, the development of an unifying
framework for data mining is listed first. One step towards developing a general framework for data mining is constructing an ontology of data mining.

In this chapter, we present our proposal for an ontology of data mining (DM) named OntoDM [35, 36]. Our ontology design takes into consideration the best practices in ontology engineering. We use an upper level ontology - Basic Formal Ontology (BFO)\(^1\) to define the upper level classes. We also use the OBO Relational Ontology (RO)\(^2\) and other related ontologies for representing scientific investigations, to define the semantics of the relationships between the data mining entities, and provide is-a completeness and single is-a inheritance for all DM entities.

The OntoDM ontology is based on a recent proposal for a general framework for data mining [13]. We have developed our ontology in the most general fashion in order to be able to represent complex data mining entities. These are becoming more and more popular in research areas such as mining structured data and constraint-based mining.

The rest of the chapter is structured as follows. In Section 2.2, we present the ontology design principles and we put the ontology in context of other ontologies for representing scientific investigations. Section 2.3 presents the ontology upper level structure, the ontological relations employed, and the division of OntoDM into logical modules. In the following section (Section 2.4) we present the basic entities in data mining, following the basic principles from the proposal of a general framework for data mining. In Section 2.5, we describe how we represent the data mining entities in all three modules of the ontology. We conclude the chapter with a critical overview of related work (Section 2.6), discussion and conclusions (Section 2.7).

### 2.2 Design Principles for the OntoDM ontology

#### 2.2.1 Motivation

The motivation for developing an ontology of data mining is multi-fold. First, the area of data mining is developing rapidly and one of the most challenging problems deals with developing a general framework for mining of structured data and constraint-based data mining. By developing an ontology of data mining we are taking one step toward solving this problem. The ontology would formalize the basic entities (e.g., dataset and data mining algorithm in data mining) and define the relations between the entities. After the basic entities are identified and logically defined, we can build upon them and define more complex entities (e.g., constraints, constraint-based data mining task, data mining query, data mining scenario and data mining experiment).

---

\(^1\) BFO: [http://www.ifomis.org/bfo](http://www.ifomis.org/bfo)

\(^2\) RO: [http://www.obofoundry.org/ro/](http://www.obofoundry.org/ro/)
Second, there exist several proposals for ontologies of data mining, but the majority of them are light-weight, aimed at covering a particular use-case in data mining, are of a limited scope, and highly use-case dependent. Most of the developments are with the aim of automatic planning of data mining workflows [2, 49, 50, 24, 22, 26, 11, 12]. Some of the developments are aimed at describing of data mining services on the GRID [8, 5]. Data mining is a domain that needs a heavy-weight ontology with a broader scope, where much attention is paid to the precise meaning of each entity, semantically rigorous relations between entities and compliance to an upper level ontology, and compatibility with ontologies for the domains of application (e.g., biology, environmental sciences).

Finally, an ontology of data mining should define what is the minimum information required for the description of a data mining investigation. Biology is leading the way in developing standards for recording and representation of scientific data and biological investigations [16] (e.g., already more than 50 journals require compliance of the reporting in papers results of microarray experiments to the Minimum Information About a Microarray Experiment - MIAME standard [14]). The researchers in the domain of data mining should follow this good practice and the ontology of data mining should support the development of standards for performing and recording of data mining investigations.

To summarize, the major goal of our ontology is to provide a structured vocabulary of entities sufficient for the description of the scientific domain of data mining. In order to achieve this goal the ontology should:

- represent the fundamental data mining entities;
- allow support for representing entities for mining structured data at all levels: the entities representing propositional (single table) data mining should be a special case (subclass) of a more general framework of mining structured data;
- be extensible, i.e., support representing complex data mining entities using fundamental data mining entities;
- use an upper level ontology and formally defined relations based on upper-level classes in order to provide connections to other domain ontologies and provide reasoning capabilities across domains;
- reuse classes and relations from other ontologies representing scientific investigations and outcomes of research and
- support the representation of data mining investigations.

### 2.2.2 OntoDM design principles

The OntoDM ontology design takes into consideration the best practices in ontology engineering. We use the upper level ontology BFO (Basic Formal Ontology)\(^3\) to define the upper level classes, We use the OBO Relational Ontology (RO)\(^4\) and an

---

3 BFO: [http://www.ifomis.org/bfo](http://www.ifomis.org/bfo)

extended set of RO relations to define the semantics of the relationships between the data mining entities: in this way, we achieve is-a completeness and single is-a inheritance for all data mining entities.

OntoDM aims to follow the OBO Foundry principles in ontology engineering that are widely accepted in the biomedical domain. The main OBO Foundry principles state that “the ontology is open and available to be used by all”, “is in a common formal language”, ”includes textual definition of all terms”, ”uses relations which are unambiguously defined”, ”is orthogonal to other OBO ontologies” and ”follows a naming convention” [39]. In this way, OntoDM is built on a sound theoretical foundation and will be compliant with other (e.g., biological) domain ontologies. Our ontology will be compatible with other formalisms, and thus widely available for sharing and reuse of already formalized knowledge.

OntoDM is ”in a common formal language”: it is expressed in OWL-DL, a de-facto standard for representing ontologies. OntoDM is being developed using the Protege ontology editor. It consists of three main components: classes, relations (a hierarchical structure of is-a relations and relations other than is-a ), and instances.

### 2.2.3 Ontologies for representing scientific investigations

Concerning the relationship to other ontologies, we note here that there exist several formalisms for describing scientific investigations and outcomes of research. Below we review five proposals that are relevant for describing data mining investigations: the Basic Formal Ontology (BFO) as an upper level ontology, the Ontology for Biomedical Investigations (OBI) [7], the Information Artifact Ontology (IAO) [8], the Ontology of Scientific Experiments (EXPO) [45] and its extension LABORS [28],and the Ontology of Experiment Actions (EXACT) [43]. In the design of the OntoDM ontology, we reuse and further extend their structure and use their philosophy to identify and organize the OntoDM entities in an is-a class hierarchy, following the MIREOT (The Minimum Information to Reference an External Ontology Term) principle [10].

**Basic Formal Ontology - BFO.** The philosophy of BFO [20] overlaps in some parts with the philosophy of other upper level ontologies, such as DOLCE (Descriptive Ontology for Linguistic and Cognitive Engineering) [19] and SUMO (Suggested Upper Merged Ontology)[34]. However, BFO is narrowly focused on the task of providing a genuine upper ontology which can be used in support of domain ontologies developed for scientific research, as for example in biomedicine. It is included within the framework of the OBO Foundry.

---


6 Protege: [http://protege.stanford.edu](http://protege.stanford.edu)

7 OBI: [http://purl.obolibrary.org/obo/obi](http://purl.obolibrary.org/obo/obi)

BFO recognizes a basic distinction between two kinds of entities: substantial entities or continuants and processual entities or occurrents. Continuants, represent entities that endure through time, while maintaining their identity. Occurrents represent entities that happen, unfold and develop in time. The characteristic feature of occurrents, or processual entities, is that they are extended both in space and time.

**Ontology of biomedical investigations - OBI.** The OBI ontology aims to provide a standard for the representation of biological and biomedical investigations. The OBI Consortium is developing a set of universal terms that are applicable across various biological and technological domains and domain specific terms relevant only to a given domain. The ontology supports consistent annotation of biomedical investigations regardless of the particular field of the study [6]. OBI defines an investigation as a process with several parts, including planning an overall study design, executing the designed study, and documenting the results.

The OBI ontology employs rigid logic and semantics as it uses an upper level ontology BFO and the RO relations to define the top classes and a set of relations. OBI defines occurrences (processes) and continuants (materials, instruments, qualities, roles, functions) relevant to biomedical domains. The Data Transformation Branch is an OBI branch with the scope of identifying and representing entities and relations to describe processes which produce output data given some input data, and the work done by this branch is directly relevant to the OntoDM ontology.

OBI is fully compliant with the existing formalisms in biomedical domains. OBI is an OBO Foundry candidate [15]. The OBO Foundry requires all member ontologies to follow the same design principles, the same set of relations, the same upper ontology, and to define a single class only once within OBO to facilitate integration and automatic reasoning.

**Information Artifact Ontology - IAO.** Due to the limitations of BFO in dealing with information, an Information Artifact Ontology (IAO) has been recently proposed as a spin-off of the OBI project. The IAO ontology aims to be a mid-level ontology, dealing with information content entities (e.g., documents, file formats, specifications), processes that consume or produce information content entities (e.g., writing, documenting, measuring), material bearers of information (e.g., books, journals) and relations in which one of the relata is an information content entity (e.g., is-about, denotes, cites). IAO is currently available only in a draft version, but we have included the most stable and relevant classes into OntoDM.

**Ontology of experiments - EXPO and LABORS.** The formal definition of experiments for analysis, annotation and sharing of results is a fundamental part of scientific practice. A generic ontology of experiments EXPO [45] tries to define the principal entities for representation of scientific investigations. EXPO defines types of investigations: EXPO:computational investigation, EXPO:physical investigation and their principal components: EXPO:investigator, EXPO:method, EXPO:result, EXPO:conclusion.

The EXPO ontology is of a general value in describing experiments from various areas of research. This was demonstrated with the use of the ontology for the
description of high-energy physics and phylogenetics investigations. The ontology uses a subset of SUMO as top classes, and a minimized set of relations in order to provide compliance with the existing formalisms.

The LABORS ontology is an extension of EXPO for the description of automated investigations (the Robot Scientist Project). LABORS defines research units, such as investigation, study, test, trial and replicate: These are required for the description of complex multilayered investigations carried out by a robot [28].

**Ontology of experiment actions - EXACT** The ontology of experiment actions (EXACT) [43] aims to provide a structured vocabulary of terms for the description of protocols in biomedical domains. The main contribution of this ontology is the formalization of biological laboratory protocols in order to enable repeatability and reuse of already published experiment protocols. This ontology and the COW (Combining Ontologies with Workflows) software tool were used as a use case to formalize laboratory protocols in the form of workflows [30].

### 2.3 OntoDM Structure and Implementation

The upper level structure of the OntoDM ontology is mapped and aligned closely to the structure of the OBI ontology, a state-of-the-art ontology for describing biomedical investigations. In order to describe informational entities, the OBI ontology uses classes from the IAO ontology. A design decision was made to include relevant classes from IAO into OntoDM for the same purpose. As both the OBI and IAO ontologies to use BFO as a top ontology, we decided use BFO top level classes to represent entities which exist in the real world. In addition, we follow the design philosophy of EXPO/LABORS to represent mathematical entities.

The OntoDM ontology aims at interoperability among the ontologies: It thus includes formally defined ontological relations, based on upper level ontology classes, in order to achieve the desired level of expressiveness and interoperability. The set of relations is composed of relations from the relational ontology (RO) [42], a relation from the EXACT ontology [43], and relations from IAO and OBI. All of the relations used are formally defined on an instance and class level.

In the remainder of this section, we present an overview of the upper level classes, and the relations used in OntoDM, and then discuss how design decisions on the structure of the ontology allow us to establish a modular ontology for representing the domain of data mining. The modular structure of the ontology is a necessity in order to represent different aspects of the data mining and knowledge discovery process and to facilitate the different needs of the potential users of the ontology.

---

9 [http://www.aber.ac.uk/compsci/Research/bio/robotsci/]
### 2.3.1 Upper level is-a hierarchy

In Figure 2.1, we present the upper level OntoDM class hierarchy. Below we give more details on the meaning of each upper level class. The upper level classes are further extended in the OntoDM ontology.

**Continuants.** An entity that exists in full at any time in which it exists at all, persists through time while maintaining its identity, and has no temporal parts in the BFO ontology is called a **BFO:continuant** (e.g., a person, a heart). A **BFO:dependent continuant** is a continuant that is either dependent on one or other independent continuant bearers or inheres in or is borne by other entities. Dependent continuants in BFO can be generically dependend or specifically dependent. A **BFO:generically dependent continuant** is a dependent continuant where every instance of A requires some instance of B, but which instance of B serves can change from time to time (e.g., a certain PDF file that exists in different and in several hard drives). For a **BFO:specifically dependent continuant**, every instance of A requires some specific instance of B which must always be the same (e.g., the role of being a doctor, the function of the heart in the body etc.).

The **IAO:information content entity** (ICE) was recently introduced into IAO (motivated by the need of OBI) and denotes all entities that are generically dependent on some artifact and stand in relation of aboutness (**is-about**) to some entity. Examples of ICE include data, narrative objects, graphs etc. The introduction of ICE enables the representation of different ways that information relates to the world, sufficient for representing scientific investigations (and in case of OBI, specifically biomedical investigations).

A **BFO: Realizable entity** (RE) is a specifically dependent continuant and includes all entities that can be executed (manifested, actualized, realized) in concrete occurrences (e.g., processes). RE are entities whose instances contain
periods of actualization, when they are manifested through processes in which their bearers participate. Examples of RE are plans, roles, functions and dispositions.

An IAO: directive informational entity\(^\text{10}\) (DIC) is an information content entity that concerns a realizable entity. DICs are information content entities whose concretizations indicate to their bearer how to realize them in a process. Examples of DICs are: objective specification, plan specification, action specification, etc. An IAO: objective specification describes an intended process endpoint. An IAO: plan specification includes parts such as: objective specification, action specifications and conditional specifications. When concretized, it is executed in a process in which the bearer tries to achieve the objectives, in part by taking the actions specified.

**Occurents.** An entity that has temporal parts and that happens, unfolds or develops through time in the BFO ontology is called an BFO: occurrent (e.g., the life of an organism). A BFO: processual entity is an occurrent that exists in time by occurring or happening, has temporal parts and always involves and depends on some entity. A BFO: process is a processual entity that is a maximally connected spatiotemporal whole and has beginnings and endings (e.g., the process of sleeping).

An OBI: planned process is a processual entity that realizes a OBI: plan which is the concretization of a IAO: plan specification in order to achieve the objectives IAO: objective specification. Process entities have as participants continuants, and participants can be also active and in that case they are called agents.

### 2.3.2 Ontological relations

Relations are the most essential part of a well designed ontology. It is thus crucial that the relations are logically defined. At every point of ontology development, from the initial conceptualization, through the construction, to its use, all the relations introduced should not change their meaning. The consistent use of rigorous definitions of formal relations is a major step toward enabling the achievement of interoperability among ontologies in the support of automated reasoning across data derived from multiple domains. The full set of relations used in the OntoDM ontology is presented in Table 2.1. Below we give a brief overview of their formal meaning.

**Fundamental relations.** The fundamental relations is-a and has-part are used to express subsumption and part-whole relationships between entities. The relation has-instance is a relation that connects a class with an instance of that class. The fundamental relations are formally defined in the Relational Ontology [42], both at class and instance level.

**Information entity relations.** We included a primitive relation from IAO (is-about) that relates an information artifact to an entity. In this ontology we reuse

---

\(^{10}\) A directive information entity, before the OBI RC1 version, was named informational entity about a realizable.
Table 2.1 Relations in OntoDM. The relations are presented with the name of the relation, the origin of the relation, the domain and range of use and the inverse relation (where defined)

<table>
<thead>
<tr>
<th>Relation</th>
<th>Origin</th>
<th>Domain</th>
<th>Range</th>
<th>Inverse relation</th>
</tr>
</thead>
<tbody>
<tr>
<td>is-a</td>
<td>RO</td>
<td>entity</td>
<td>entity</td>
<td>sub-class-of</td>
</tr>
<tr>
<td>has-part</td>
<td>RO</td>
<td>entity</td>
<td>entity</td>
<td>part-of</td>
</tr>
<tr>
<td>has-instance</td>
<td>RO</td>
<td>instance</td>
<td>instance</td>
<td>instance-of</td>
</tr>
<tr>
<td>has-participant</td>
<td>RO</td>
<td>BFO:occurent</td>
<td>BFO:continuant</td>
<td>participates-in</td>
</tr>
<tr>
<td>has-agent</td>
<td>RO</td>
<td>BFO:entity</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ts-about</td>
<td>IAO</td>
<td>IAO:information entity</td>
<td>BFO:entity</td>
<td></td>
</tr>
<tr>
<td>has-information</td>
<td>EXACT</td>
<td>agent of a process</td>
<td>IAO:information content entity</td>
<td></td>
</tr>
<tr>
<td>has-specified input</td>
<td>OBI</td>
<td>BFO:processual entity</td>
<td>BFO:dependent continuant</td>
<td>is-specified input-of</td>
</tr>
<tr>
<td>has-specified output</td>
<td>OBI</td>
<td>BFO:processual entity</td>
<td>BFO:dependent continuant</td>
<td>is-specified output-of</td>
</tr>
<tr>
<td>inheres-in</td>
<td>OBI</td>
<td>BFO:dependent continuant</td>
<td>BFO:continuant</td>
<td>bearer-of</td>
</tr>
<tr>
<td>is-concretization-of</td>
<td>OBI</td>
<td>BFO:specifically dependent continuant</td>
<td>BFO:generically dependent continuant</td>
<td>is-concretized-as</td>
</tr>
<tr>
<td>realizes</td>
<td>OBI</td>
<td>BFO:process</td>
<td>BFO:realizable entity</td>
<td>is-realized-by</td>
</tr>
<tr>
<td>achieves-planned-objective</td>
<td>OBI</td>
<td>OBI:planned process</td>
<td>IAO:objective specification</td>
<td>achieved-by</td>
</tr>
</tbody>
</table>

the relation has-information defined in the EXACT ontology [43] to relate an agent of a process to a certain portion of information (information entity) that is essential for participating in the process.

Process relations. The relations has-participant and has-agent (both defined in RO) express the relationship between a process and participants in a process, that can be passive or active (in case of agents). The relations has-specified-input and has-specified-output have been recently introduced into the OBI ontology and are candidate relations for RO. These relations are specializations of the relation has-participant, and are used for relating a process with special types of participants, inputs and outputs of the process. We made a design decision to include them in OntoDM in order to increase the expressiveness and interoperability with the OBI ontology.

Role and quality relations. The relation between a dependent continuant and an entity is expressed via the relation inheres-in (defined in the OBI ontology and candidate for inclusion into RO). This relation links qualities, roles, functions, dispositions and other dependent continuants to their bearers. It is a super-relation of the relations role-of and quality-of.

Relations between information entities, realizable entities and processes. The relation is-concretization-of (introduced by the IAO ontology) expresses the relationship between a generically dependent continuant (GDC) and a specifically dependent continuant (SCD). In the OBI ontology, this relation is defined in the following way: "A GDC may inhere in more than one entity. It does so by virtue
of the fact that there is, for each entity that it inheres, a specifically dependent ‘concretization’ of the GDC that is specifically dependent”.

The relation \textit{realizes} is used to express the relation between a process and a function (realizable entity), where the unfolding of the process requires execution of a function (execution of the realizable entity). The relation \textit{achieves-planned-objective} links a planned process with its planned objectives. The planned process realizes a plan which is a concretization of a plan specification, which has as a part an objective specification. The objectives listed in the objective specification are met at the end of the planned process. Both relations were introduced by the OBI ontology.

2.3.3 \textit{Modularity: Specification, implementation, application}

In Figure 2.3.3, we present three modules of the ontology capable of describing three different aspects of data mining. The first module, named ”specification”, is aimed to contain and represent the informational entities in data mining. Examples of such entities are: data mining task, algorithm specification, dataset description, generalization specification etc. The second module, named ”implementation”, is aimed to describe concrete implementations of algorithms, implementations of components of algorithms, such as distance functions and generalizations produced by the mining process. The third module, named ”application”, aims at describing the data mining process and the participants of the process in the context of data mining scenarios. Example of processual entities are: the application of an algorithm

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{ontology_diagram.png}
\caption{Three levels of description: specification, implementation and application. The rectangle objects in the figure represent ontology classes. The ontological relations are represented with directed labeled arrows. The relations that do not have an attached label are \textit{is-a} relations.}
\end{figure}
implementation (execution of an algorithm) and the application of a predictive model on new data etc.

The modules are inter connected with the previously introduced relations. In that fashion, a specification is-concretized-as an implementation. Next, an implementation is-realized-by an application. Finally, an application achieves-planned-objective specification.

It is necessary to have all three aspects represented separately in the ontology as they have distinctly different nature. This will facilitate different usage of the ontology. For example, the specification aspect can be used to reason about components of data mining algorithms; the implementation aspect can be used for search over implementations of data mining algorithms and to compare various implementations and the application aspect can be used for constructing data mining scenarios and workflows, definition of participants of workflows and its parts.

2.4 Identification of Data Mining Entities

One of the first major steps in domain ontology construction is the identification of domain terms. In the case of OntoDM, we performed the identification following the principles from a proposal for general framework for data mining [13]. This enables us to have a general ontology of data mining, that can cover different aspects of the data mining domain and allow easy extensions of the ontology with new entities in a principled way. From the framework proposal, we identified a set of basic terms of data mining that are used to construct the basic data mining entities that form the core of our ontology.

The identified terms are used to describe different dimensions of data mining. These are all orthogonal dimensions and different combinations among these should be facilitated. Through combination of these basic terms and other support terms already defined in related ontologies such as BFO, IAO, OBI, EXPO/LABORS, EXACT one should be able to describe, with principled extensions of the ontology, most of the diversity present in data mining approaches today. In the remainder of this section, we present an overview of the general framework for (structured) data mining [13], describing first the basic principles of the framework, followed by an overview of basic entities such as data, generalizations, data mining task and data mining algorithms.

2.4.1 A general framework for data mining: Basic principles

One of the main 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 (in most cases structured) data are receiving and increasing amount 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 mining tasks have been considered so far within the field of data mining. By far the most common is the task of predictive modeling, 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 and others, but it is not clear whether these are of fundamental nature or can be defined by composing some of the tasks listed above. The task of a general framework for data mining would be to define the fundamental (basic) data mining tasks and allow definition of more complex tasks by combining the fundamental ones.

Finally, different types of generalizations (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).

2.4.2 Data

Data is the most basic data mining entity. A data mining algorithm takes as input a set of data (dataset). An individual datum (data example) in the dataset 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 examples are homogeneous (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. 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. It is of crucial importance to be able to deal with structured data, as these are attracting an increasing 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 $\text{Sequence}(T)$, which denotes a sequence of objects of type $T$, or $\text{LabeledGraph}(VL, EL)$, 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 $\text{Sequence}(\text{Discrete} \{\text{A, C, G, T}\})$, 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).

2.4.3 Generalizations

Generalization is a broad term that denotes the output of different data mining tasks, such as pattern mining, predictive modeling and clustering. Generalizations include probability distributions, patterns (in the sense of frequent patterns) and global models (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.

Generalizations inherently have a dual nature. They can be treated as data structures and as such represented, stored and manipulated. On the other hand, they are functions that take as input data points and map them to probabilities (in the case of probability distributions), class predictions (in the case of predictive models), cluster assignments (in the case of clusterings), or Booleans (in the case of local patterns).

The remainder of this sub-section, we first list the fundamental types of generalizations in data mining, then describe classes of generalizations (that refer to the data structure nature) and finally we describe interpreters of generalizations (that refer to the function nature).

**Fundamental types of generalizations.** Fundamental types of generalizations include: probability distributions, patterns, predictive models and clusterings.

A probability distribution $D$ on type $T$ is a mapping from objects of type $T$ to non-negative Reals, i.e., has the signature $d :: T \rightarrow R^{0+}$. 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 $p :: T \rightarrow \text{bool}$. A pattern on type $T$ is true or false on an object of type $T$. A pattern is defined as a statement (expression) in a given language, that describes (relationships among) the facts in (a subset of) the data [17].

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 $m :: T_d \rightarrow T_c$. Most often, predictive modelling is concerned with classification, where $T_c$ would be Boolean (for binary classification), 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 clustering $C$ on a set of objects $S$ of type $T$ is a function from $S$ to $\{1,\ldots,k\}$, 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 $T \rightarrow (\{1,\ldots,k\} \rightarrow \mathbb{R}^{0+})$. In hierarchical clustering, in addition to the function $C$, we get a hierarchy on top of the set $1,\ldots,k$.

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 $M: \{1,\ldots,k\} \rightarrow (T_d \rightarrow T_c)$. Performing the function composition of $M$ and $C$, i.e., applying first $C$ and then $M$, we get a predictive model on $T$.

Classes of Generalizations. Many different kinds of generalizations 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 generalization classes.

A class of generalizations $C_G$ on a set on a datatype $T$ is a set of generalizations on $T$ expressed in a language $L_G$. For each specific type of generalization we can define a specific generalization class. The languages $L_G$ refer to the data part of the generalizations. They essentially define data types for representing the generalizations. For example, 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 $L_M$.

Interpreters. There is usually a unique mapping from the data part of a generalization to the function part. This takes the data part of a generalization as input, and returns the corresponding function as an output. This mapping can be realized through a so-called interpreter. The interpreter is crucial for the semantics of a class of generalizations: a class of generalizations is only completely defined when the corresponding interpreter is defined (e.g., interpreter for models $I_M$ is part of the definition of the class $C_M$).

For illustration, given a data type $T$, an example $E$ of type $T$, and a pattern $P$ of type $p :: T \rightarrow \text{bool}$, an interpreter $I$ returns the result of applying $P$ to $E$, i.e., $I(P,E) = P(E)$. The signature of the interpreter is $i :: p \rightarrow T \rightarrow \text{bool}$. If we apply the interpreter to a pattern and an example, we obtain a Boolean value.

2.4.4 Data mining task

In essence, the task of data mining is to produce a generalization from a given set of data. Here we will focus on four fundamental tasks, according to the generalizations produced: estimating the (joint) probability distribution, learning predictive models, clustering and finding valid (frequent) patterns.
Estimating the (Joint) Probability Distribution. Probably the most general data mining task [21] is the task of estimating the (joint) probability distribution \( D \) over type \( T \) from a set of data examples or a sample drawn from that distribution.

Learning a 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, \( m : T_d \rightarrow T_c \), 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.

Clustering The task of clustering in general is concerned with grouping objects into classes of similar objects [25]. Given a set of examples (object descriptions), the task of clustering is to partition these examples into subsets, called clusters. 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).

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 \( \{\text{bread, butter}\} \)), which are often found together in a transaction (e.g., a market basket) [1].

2.4.5 Data mining algorithms

A data mining algorithm is an algorithm (implemented in a computer program), designed to solve a data mining task. It takes as input a dataset of examples of a given datatype and produces as output a generalization (from a given class) on the given datatype. A data mining algorithm can typically handle examples of a limited set (class) of datatypes: For example, a rule learning algorithm might handle only tuples of Boolean attributes and a boolean class.

Just as we have classes of datatypes, classes of generalizations and data mining tasks, we have classes of data mining algorithms. The latter are directly related to the input and output of the algorithm, but can depend also on the specifics of the algorithm, such as the basic components of the algorithm (e.g., heuristic function, search method). For example, for the class of decision tree building algorithms, we can have two subclasses corresponding to top-down induction and beam-search (cf. Chapter 7) of this volume.

As stated earlier in this chapter, a very desirable property of a data mining framework is to treat the mining of different types of structured data in a uniform fashion. In this context, data mining algorithms should be able to handle as broad classes of datatypes at the input as possible. We will refer to algorithms that can
handle arbitrary types of structured data at the input as generic. Generic data mining algorithms would typically have as parameters some of their components, e.g., a heuristic function in decision tree induction or a distance in distance-based clustering.

The general framework for data mining proposed by Džeroski [13] discusses several types of data mining algorithms and components thereof. The basic components include distances, features, kernels and generality/refinement operators. The framework proposes that the components of data mining should be treated as first-class citizens in inductive databases, much like generalizations (including patterns and models). We follow this approach and represent the entities corresponding to algorithm components in OntoDM: We thus give a brief overview thereof below.

**Distances.** The major components of distance-based algorithms are distance and prototype functions. A distance function \( d \) for type \( T \) is a mapping from pairs of objects of type \( T \) to non-negative reals: \( d : : T \times T \rightarrow R^{0+} \). Distances are of crucial importance for clustering and predictive modelling. In clusters, we want to minimize the distance between objects in a cluster. In predictive modelling, we need to compare the true value of the target to the predicted one, for any given example. This is typically done by finding their distance.

A prototype is a representative of all the objects in a given set \( S \). In the context of a given distance \( d \), this is the object \( o \) that has the lowest average square distance to all of the objects in \( S \). 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: \( p : : \text{Set}(T) \rightarrow T \).

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. 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) 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 (\( k \)-NN), 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. In the 1-NN case, we do not need this prototype function, as the prediction is simply copied from the nearest neighbor.

**Features and feature based representation.** Most of data mining algorithms use a feature based representation. 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.

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 identify 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 $d$. The other two ways are related to background knowledge concerning the structure of the object or concerning domain knowledge.

**Kernels and Kernel Based Algorithms.** Technically, a kernel $k$ corresponds to the inner product in some feature space. The computational attractiveness of kernel methods[40] (KM) 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*.

KMs in general 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.

At the conceptual level, kernels elegantly relate to both features and distances. 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 [18]. There are also many algorithms capable of operating with kernels, and the most well known of which are SVMs (Support Vector Machines).

**Refinement Orders and Search of Generalization Space.** The notion of generality is a key notion in data mining, in particular for the task of pattern discovery. To find generalizations valid in the data, data mining algorithms search the *space of generalizations* defined by the class of generalizations considered, possibly additionally restricted by constraints. To make the search efficient, the space of generalizations is typically ordered by a *generality or subsumption relation*.

The generality relation typically refers to the function part of a generalization. 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 (e.g., adds one item to an itemset).

The prototypical algorithm for mining frequent patterns starts its search with the empty pattern (set/sequence/graph), which is always frequent. It then proceeds level-wise, 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 as no refinement of an infrequent pattern can be frequent.
2.4.6 OntoDM modeling issues

The identification of domain terms is just the first step in the construction of a domain ontology. Next, there is a need to revise the terms in the sense of ontology design principles and form ontological entities. In this phase, one has to form ontological classes, represent them with their unique characteristics (called properties), relate them to other classes using ontological relations, and place them adequately in the is-a hierarchy of classes.

An identified term is not always automatically mapped to an ontological class. Often a manual adjustment by an ontology engineer is required. For example, the term “data mining algorithm” can be used in three conceptually different aspects, which should be modeled separately in the ontology.

The first aspect is a specification of the algorithm. Here an algorithm would be described with the specification of the inputs and outputs, types of generalizations produced, data mining tasks it solves, the components of algorithms, parameters that can be tuned, the publication where the algorithm has been published etc. The second aspect is a concrete implementation of the algorithm. Here we have concrete executable version of the algorithm, and several different implementations can exist based on the same specification. Finally, a third aspect is the application of an algorithm implementation to a concrete dataset and the production of an output generalization. Here we deal with the data mining process, where essential entities are the participants in the process, the sub-processes, and how the sub-processes are connected between each other (which sub-process precedes the other) etc.

The same can be exemplified with other entities in the ontology. Let us take, for example, a predictive model. The first aspect of a predictive model is its specification. Here we describe general characteristics of the model, what tasks they are produced from, model structure, parameters of the model structure, the language in which they are expressed (e.g., language of decision trees). The second aspect is a concrete (instantiated) model which is the result of execution of an algorithm implementation (a process) on a dataset. Here the instantiated model has a link to the dataset that produced it, the process that produced it, the quality measure instantiations on the data from which the model was produced etc. The final aspect is the execution of the model on new data, which is itself a process with the goal prediction. The inputs of the process are the model and the new data; the outputs are the predictions and the evaluation measures calculated.

Another important aspect in modeling the terms into an ontology is the treatment of the roles of entities. When modeling, one should define an entity with its purest properties that would allow us to differentiate it from other entities. But to do this, one has to abstract the entity from different contexts where the entity can appear. Modeling of realizations of an entity in different contexts should be done via roles of entities [33]. A typical example of a role in data mining is an operator. An operator is a role of an implementation of a data mining algorithm in the context of data mining workflows.
2.5 Representing Data Mining Entities in OntoDM

In this section, we report how the data mining entities discussed above are represented in the OntoDM ontology. Furthermore, we give an overview and examples of classes, relations and instances from the specification, implementation and application module of the ontology. In addition, we provide a discussion of the advantages of the chosen ontology design patterns.

2.5.1 Specification entities in OntoDM

One of the main goals of the OntoDM ontology is to represent entities for structured data mining. Our design decisions allow us to treat the traditional single-table data mining as a special case of structured data mining. Furthermore, the goal is to keep the design as general as possible, in order to allow easy extensions covering further new developments in the domain of data mining.

The specification module of OntoDM contains specification entities (classes and instances) for the domain of data mining. Examples of entities are datatype, dataset, generalization specification, data mining task specification and data mining algorithm specification. The specification classes are extensions of the information content entity class.

![Taxonomy and part-whole relations between basic data mining specification classes in OntoDM. The rectangle objects in the figure represent ontology classes. The ontological relations are represented with directed labeled arrows. The relations that do not have an attached label are is-a relations.](image-url)
In Figure 2.3, we present the is-a hierarchy and the part-whole relations between the basic data mining entity classes of the specification module. The most fundamental specification class in OntoDM is the datatype (See Section 2.4.2 for more details). Next, we have the datatype spec. related to the datatype through the is-about relation. The datatype spec. has two subclasses at the first level: input datatype spec. and output datatype spec.. They are used to differentiate between input and output datatypes in the formal representation of generalizations.

A generalization spec. has as its parts a datatype spec. and generalization language spec.. It is further sub-classed at the first level with the following classes: local pattern spec., global model spec. and probability distribution spec.. Having a generalization language spec. as a part of gives us the opportunity to further develop the taxonomy of generalizations by forming classes of generalizations (as discussed in Section 2.4.3).

Next, we have the data mining task spec. which is directly related to the types of generalizations via a has-part relation. This class is a subclass of IAO:objective specification. It is further sub-classed with the basic data mining tasks (See Section 2.4.4): local pattern discovery task, predictive modeling task, clustering task and probability distribution estimation task.

Finally, a data mining algorithm spec. has as its parts a data mining task spec. and data mining algorithm component spec. (See Section 2.4.5). A data mining algorithm spec. is a sub-class of IAO:plan specification and this is aligned with the IAO and OBI ontology structure, that is a IAO:plan specification has as its part IAO:objective specification.

The main advantage of having such a structure of classes (conected via has-part chains) is the ability to use the transitivity property of the has-part relation. For example, when we have an instance of data mining algorithm spec., we can use reasoning to extract the data mining task, which is an objective of the algorithm, the type of generalization the algorithm gives at its output and the datatype specification on the input and output side.

In the remaining of this subsection we will discuss in more detail the datatype entity and the representation of structured datatypes and example of instances of structured datatypes.

**Datatype.** Figure 2.4 depicts the representation of datatypes in OntoDM. A datatype can be a primitive datatype or a structured datatype (See Figure 2.4c). According to [32], a primitive datatype is “a datatype whose values are regarded fundamental - not subject to any reduction”. Primitive types can be non-ordered (e.g., discrete datatype) and ordered (e.g., inst:real datatype, inst:integer datatype). Furthermore, ordered datatypes can also be ordinal (e.g., inst:boolean datatype).

A structured datatype (or aggregated datatype in [32]) is “one whose values are made up of a number, in general more than one, of component values, each of which is a value of another datatype”. A structured datatype has two parts: datatype component spec. and aggregate datatype constructor spec.. The datatype component spec. specifies the components of the structured datatype and aggregate datatype constructor spec. specifies the datatype constructor used to compose the structure.
Fig. 2.4 Datatype specification in OntoDM: a) taxonomy of type constructors; b) structured datatype entity; c) taxonomy of datatypes. The rectangle objects in the figure represent ontology classes. The oval objects represent ontology instances. The ontological relations are represented with directed labeled arrows. The relations that don’t have an attached label are is-a relations.

Providing an adequate and complete taxonomy of datatypes is a very challenging task. In the implementation of the OntoDM ontology, we decided to follow the guidelines from [32] to represent the datatype entity and construct a taxonomy of datatypes applicable to the domain of data mining. The construction is done in a
general fashion that allows extensions in case new datatypes appear and are not covered so far. The taxonomy of datatypes is given in Figure 2.4c.

**Datatype constructor.** The taxonomy of the structured datatypes is based on the taxonomy of datatype constructors (See Figure 2.4a). A *datatype constructor* spec. can be non-aggregate or aggregate. A *non-aggregate datatype constr.* is defined in [32] as “datatypes that are produced from other datatypes by the methods familiar from languages that include them” (e.g., pointers, procedures, choices). An *aggregate datatype constr.* defines the aggregate that is used to combine the component datatypes. The aggregate type constructors classes can be further extended using different properties (e.g., ordering of components, homogeneity, how the components are distinguished - tagging or keying etc).

In this ontology, we distinguish between non-ordered and ordered aggregates. Non-ordered aggregate constructors (or bags) include: sets, tuples (or records) and undirected labeled graph. A *set constr.* is a constructor that does not allow duplicates. A *tuple constr.* is an aggregate where each component can be tagged.

A *sequence constr.* is the simplest ordered aggregate with a strict and unique ordering of the components. A *vector constr.* is a *sequence constr.*, where components are indexed and the ordering of the index induces the ordering of the components. In a similar way, using properties of aggregates, we can define other aggregates like *directed labeled graph constr.* and its subclasses *labeled tree constr.* and *labeled DAG constr.*. All defined aggregates can be further sub-classed using constraints such as homogeneity, size (number of components), etc.

**How do we define an instance of a structured datatype?** Having the representation of a datatype and datatype constructor we can represent arbitrary datatype instances. In Fig. 2.5a, we show how to represent *inst:tuple(boolean,real)*. It is an instance of the *tuple of primitive datatypes* class. *inst:tuple(boolean,real)* has two primitive datatype components (boolean and real) and a two element tuple.

---

**Fig. 2.5** An example of the structured datatype instances: a) The tuple(boolean,real) instance; b) The set{tuple(boolean,real)} instance. Dashed lines represent instance level relations and full lines represent class level relations.
constructor. In Fig. 2.5b, we show how we can construct a more complex structured datatype using previously defined instances. \( \text{inst:\{tuple(boolean,real)\}} \) has one component datatype (tuple(boolean,real)) and a homogeneous set constructor.

**Dataset.** Once we have representation of datatypes, we can represent datasets. A dataset is a IAO:information content entity and has as part data example. A dataset spec. is an information entity about a dataset, connected via the is-about relation. It has as its part a datatype specification, allowing us to have a classification of datasets using only datatype as a classification criteria.

This class can be further sub-classed with unlabeled dataset spec. class that has only input datatype specification as its part. We can further extend it with a special cases of unlabeled datasets: unlabeled propositional dataset spec. class, where the input specification is a tuple of primitives and transactional dataset spec. class where the input specification is a set of discrete. A labeled dataset spec. is a specialization of unlabeled dataset spec. class, where we have additionally defined output datatype specification.

![Fig. 2.6 The dataset entity in OntoDM. The ontological relations are represented with directed labeled arrows. The relations that don’t have an attached label are is-a relations.](image)

### 2.5.2 Implementation and application entities in OntoDM

In the previous subsection, we gave an overview of the specification module of the OntoDM ontology. The specification entities are connected via relations to their "counter part" entities in the implementation and application modules. In this subsection, we briefly describe the two modules and give an illustrative example how the three modules are interconnected, presenting example instances of classes on all three levels.

**Implementation entities.** Entities in the implementation module include implementations of algorithms, functions, instantiations of predictive models
Fig. 2.7 An example of the connection between the three modules in OntoDM ontology: specification, implementation and application. The example shows defined instances of classes on all three levels. The rectangle boxes represent ontology classes. The oval boxes represent instances of classes. Dashed lines represent instance level relations, while the full lines represent class level relations. Relations that are not labeled are is-a relations.

resulting from the application of a data mining algorithm implementation on a concrete dataset. All classes are extensions of BFO:realizable entity (see Figure 2.7). A data mining algorithm implementation is an extension of OBI:plan. A generalization is an extension of the mapping class. The connection with
the specification module is made through the relation: an implementation is-concretization-of specification.

**Application entities.** Entities in the application module are all extensions of OBI: planned process (See Figure 2.7). It contains entities representing parts of the knowledge discovery process, such as execution of DM algorithm implementation and execution of predictive model. The execution of DM algorithm implementation is a realization (linked with realizes) of an DM algorithm implementation. Since the execution of an algorithm is a planned process it has input (dataset), an output (generalization) and achieves the planned objective data mining task.

**Illustrative example.** In Figure 2.7, we present example instances in the OntoDM ontology. The instances are represented as oval objects and the relations between instances are marked with dashed lines. In this example, we are representing the clus-HMC algorithm in all three modules. clus-HMC in an algorithm for predicting structured outputs: it learns decision trees for hierarchical multi-label classification [48].

The inst:clus-HMC process ID0001 is an instance of a predictive modeling process. It has as its input a inst:HMC dataset ID0001 and as its output inst:clus-HMC decision tree ID0001. The inst:HMC dataset ID0001 is an instance of the dataset class and is connected to the HMC dataset ID0001 spec. via the is-about relation. The dataset specification contains the input and output datatypes of the dataset (inst:input tuple of primitives and inst:output DAG). The inst:clus-HMC decision tree ID0001 is a concretization of inst:clus-HMC decision tree spec. and is realized by a inst:clus-HMC decision tree execution process in the case we want to obtain predictions for new examples.

The inst:clus-HMC process ID0001 realizes the inst:clus-HMC algorithm implementation, which is a concretization of the clus-HMC algorithm specification. The process achieves the planned objective inst:HMC learning task, which is an instance of the decision tree learning task class.

### 2.6 Related Work

The main developments in formal representation of data mining entities in the form of ontologies take place in the domain of data mining workflow construction, data mining services, and describing data mining resources on the GRID. Other research in ontologies for data mining include formal representations of machine learning experiments in context of experiment databases. Finally, there is an increasing interest in extracting data mining entities from the data mining literature. In the remainder of this section, we briefly summarize the contributions in all these domains.

**Data mining workflows.** A prototype of an Intelligent Discovery Assistant (IDA) has been proposed [2], which provides users with systematic enumerations of
valid sequences of data mining operators (called data mining processes). Effective rankings of the processes by different criteria are also provided in order to facilitate the choice of data mining processes to execute or solve a concrete data mining task. This automated system takes an advantage of an explicit ontology of data mining operators (algorithms). A light-weight ontology is used that contains only a hierarchy of data mining operators divided into three main classes: preprocessing operators, induction algorithms and post processing operators. The leaves of the hierarchy are the actual operators. The ontology does not contain information about the internal structure of the operators and the taxonomy is produced only according to the role that the operator has in the knowledge discovery process.

Building upon this work has been proposed [24] in a proposal of an intelligent data mining assistant that combines planning and meta-learning for automatic design of data mining workflows. A knowledge driven planner relies on a knowledge discovery ontology [2], to determine the valid set of operators for each step in the workflow. A probabilistic meta-learner is proposed for selecting the most appropriate operators by using relational similarity measures and kernel functions.

The problem of semi-automatic design of workflows for complex knowledge discovery tasks has also been addressed by Žakova et al. [49, 50]. The idea is to automatically propose workflows for the given type of inputs and required outputs of the discovery process. This is done by formalizing the notions of a knowledge type and data mining algorithm in the form of an ontology (named KD ontology). The planning algorithm accepts task descriptions expressed using the vocabulary of the ontology.

Kietz et al. [26, 27] present a data mining ontology for workflow planning. The ontology is designed to contain all the information necessary to support a 3rd generation KDD Support System. This includes the objects manipulated by the system, the meta data needed, the operators (i.e., algorithms) used and a goal description. The vocabulary of the ontology is used further for Hierarchical Task Network planning (HTN).

Hilario et al. [22] present their vision of a data mining ontology designed to support meta-learning for algorithm and model selection in the context of data mining workflow optimization. The ontology (named DMOP) is viewed as the repository of the intelligent assistant’s data mining expertise, containing representations of data mining tasks, algorithms and models.

Diamantini and Potena [11] introduce a semantic based, service oriented framework for tools sharing and reuse, in order to give support for the semantic enrichment through semantic annotation of KDD (Knowledge Discovery in Databases) tools and deployment of tools as web services. For describing the domain, they propose an ontology named KDDONTO [12] which is developed having in mind the central role of a KDD algorithm and their composition (similar to the work presented in [2, 49]).

**GRID.** In the context of GRID programming, Cannataro and Comito [8] propose a design and implementation of an ontology of data mining. The motivation for building the ontology comes from the context of the author’s work in Knowledge
GRID [9]. The main goals of the ontology are to allow the semantic search of data mining software and other data mining resources and to assist the user by suggesting the software to use on the basis of the user’s requirements and needs. The proposed DAMON (DAta Mining ONtology) ontology is built through a characterization of available data mining software.

Brezany et al. [5] introduce an ontology-based framework for automated construction of complex interactive data mining workflows as a means of improving productivity of GRID-enabled data systems. For this purpose they develop a data mining ontology which is based on concepts from industry standards such as: the predictive model mark-up language (PMML) 11, WEKA [51] and the Java data mining API [23].

**Experiment databases.** As data mining and machine learning are experimental sciences, insight into the performance of a particular algorithm is obtained by implementing it and studying how it behaves on different datasets. Blockeel and Vanschoren [3, 4] (also Vanschoren and Blockeel in this volume) propose an experimental methodology based on experiment database in order to allow repeatability of experiments and generalizability of experimental results in machine learning.

Vanschoren et al. [46] propose an XML based language (named ExpML) for describing classification and regression experiments. In this process, the authors identified the main entities for formalizing a representation of machine learning experiments and implemented it in an ontology (named Exposé) [47]. This ontology is based on the same design principles as the OntoDM ontology, presented in this chapter, and further uses and extends some of the OntoDM classes.

**Identification of entities from literature.** Peng et al. [37] survey a large collection of data mining and knowledge discovery literature in order to identify and classify the data mining entities into high-level categories using grounded theory approach and validating the classification using document clustering. As a result of the study the authors have identified eight main areas of data mining and knowledge discovery: data mining tasks, learning methods and tasks, mining complex data, foundations of data mining, data mining software and systems, high-performance and distributed data mining, data mining applications and data mining process and project.

### 2.7 Conclusion

In this chapter, we have presented the OntoDM ontology for data mining, based on a recent proposal for a general framework of data mining. OntoDM is developed as a heavy-weight ontology of the data mining, starting from first principles as laid out by the framework, and including a significant amount of detail on basic data

---

mining entities. Entities represented in OntoDM include data (datatypes, datasets), data mining tasks (e.g., predictive modeling, clustering), data mining algorithms and their components, and generalizations (e.g., patterns and models output by data mining algorithms).

OntoDM is very general and allows us to represent much of the diversity in data mining research, including recently developed approaches. For example, OntoDM covers the area of mining structured data, including both the mining of frequent patterns from structured data and the prediction of structured outputs. Also, entities from the area of constraint-based data mining and inductive databases are included, such as evaluation functions, constraints, and data mining scenarios.

In the design of OntoDM, we have followed best practices in ontology engineering. We reuse upper-level ontology categories and well-defined ontological relations accepted widely in other ontologies for representing scientific investigations. Using these design principles we can link the OntoDM ontology to other domain ontologies (e.g., ontologies developed under the OBO Foundry) and provides reasoning capabilities across domains. The ontology is divided into three logical modules (specification, implementation, application).

Consequently, OntoDM can be used to support a broad variety of tasks. For example, it can be used to search for different implementations of an algorithm, to support the composition of data mining algorithms from reusable components, as well as the construction of data mining scenarios and workflows. It can also be used for representing and annotating data mining investigations.

We are currently working on the further development of several aspects of the ontology, such as the taxonomies of generalizations, tasks and algorithms. Some of these will require further development and extension of the general framework for data mining that we have used a starting point (concerning, e.g., the more precise representation of DM algorithm components). Next, we plan to populate the ontology with specific instances of the present classes. Furthermore, we plan to connect the OntoDM ontology with ontologies of application domains (e.g., The Ontology for Drug Discovery Investigations [38]) by developing application specific use cases. Finally, applying the OntoDM design principles on the development of ontologies for other areas of computer science, is one of the most important long term objectives of our research.

Availability. The OntoDM ontology is available at: http://kt.ijs.si/pance_panov/OntoDM/

Acknowledgements Part of the research presented in this chapter was conducted within the project IQ (Inductive Queries for mining patterns and models) funded by the European Commission of the EU under contract number FP6-IST 516169. Panče Panov and Sašo Džeroski are currently supported by the Slovenian Research Agency through the research projects Advanced machine learning methods for automated modelling of dynamic systems (under grant J2-0734) and Data Mining for Integrative Data Analysis in Systems Biology (under grant J2-2285). For a complete list of agencies, grants and institutions currently supporting Sašo Džeroski, please consult the Acknowledgements chapter of this volume.
References


Inductive Databases and Constraint-Based Data Mining
Dzeroski, S.; Goethals, B.; Panov, P. (Eds.)
2010, XVII, 456 p., Hardcover