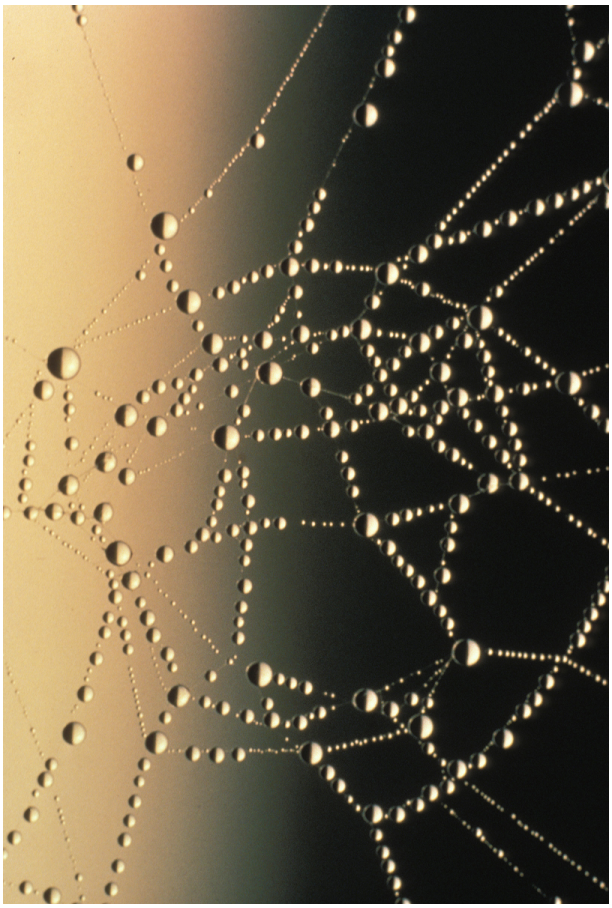
University of Zurich
Department of Informatics

Dynamic and Distributed
Information Systems

SPARQL-ML: Knowledge Discovery for the Semantic Web



Diploma Thesis December 18, 2007

André Locher
of Bratsch VS, Switzerland

Student-ID: 03-706-405
andre@outerlimits.ch

Advisor: **Christoph Kiefer**

Prof. Abraham Bernstein, PhD
Department of Informatics
University of Zurich
http://www.ifi.uzh.ch/ddis

Acknowledgements

I would like to thank Christoph Kiefer and Prof. Abraham Bernstein for giving me the opportunity to write this thesis and for their valuable input, which always made me strive for more. Additionally I want to give a big thank you to Claudia von Bastian for her moral support during the last six months and of course for proofreading. I would also like to use this opportunity to thank my parents for supporting my studies and allowing me to go all the way.

Abstract

Machine learning as well as data mining has been successfully applied to automatically or semi-automatically create Semantic Web data from plain data. Only little work has been done so far to explore the possibilities of machine learning to induce models from existing Semantic Web data. The interlinked structure of Semantic Web data allows to include relations between entities in addition to attributes of entities of propositional data mining techniques. It is, therefore, a perfect match for Statistical Relational Learning methods (SRL), which combine relational learning with statistics and probability theory.

This thesis presents SPARQL-ML, a novel approach to perform data mining tasks for knowledge discovery in the Semantic Web. Our approach is based on SPARQL and allows the use of statistical relational learning methods, such as Relational Probability Trees and Relational Bayesian Classifiers, as well as traditional propositional learning methods. We perform different experiments to evaluate our approach on synthetic and real-world datasets. The results show that SPARQL-ML is able to successfully combine statistical induction and logic deduction.

Zusammenfassung

Schon heute wird Machine Learning und Data Mining erfolgreich für das automatische und semi-automatische Erstellen von Semantic Web Daten eingesetzt. Vergleichsweise wenig wurden hingegen die Möglichkeiten des Machine Learnings in Bezug auf das Erlernen von Modellen existierender Semantic Web Daten erforscht. Durch die verkettete Struktur der Semantic Web Daten ist es möglich, neben propositionalen Attributen von Entitäten auch die Beziehungen zwischen diesen in das Data Mining einzubeziehen. Es sind also optimale Voraussetzungen für die Verwendung der Methoden des Statistical Relational Learnings gegeben, das das Lernen von Beziehungen mit Statistik und Wahrscheinlichkeitstheorie verknüpft.

In der vorliegenden Arbeit wird SPARQL-ML vorgestellt, ein neuartiger Ansatz, um Data Mining auf Semantic Web Daten anzuwenden. Er basiert auf SPARQL und ermöglicht die Verwendung von Statistical Relational Learning Methoden wie beispielsweise Relational Probability Trees und Relational Bayesian Classifiers, aber auch traditionelle propositionale Lernmethoden. Zur Evaluation dieses Ansatzes wurden verschiedene Experimente mit künstlichen und realen Datensätzen durchgeführt. Die Resultate zeigen, dass SPARQL-ML die Vorteile statistischer Induktion und logischer Deduktion erfolgreich vereinen kann.

Table of Contents

1

Introduction

1.1 Motivation

With the large and continuously growing amount of interlinked Semantic Web data, it is more and more discussed, how machine learning and data mining can be applied for this data. Machine learning techniques have been successfully used to automatically or semi-automatically create Semantic Web data from plain data. However, only little work has been done so far to answer the question on how we can best apply machine learning techniques to learn from existing Semantic Web data. Learning algorithms should be able to exploit the relational structure of the Semantic Web data by not only taking the intrinsic attributes of objects, for which a prediction should be made, but also the extrinsic relations to other objects into account. It has been agreed in the machine learning/data mining community that models trained from relational data should perform at least as well as models without taking into account this additional information. In addition, it has been shown that Statistical Relational Learning (SRL) methods can induce models without prior propositionalization (*i.e.*, translation to a single table) of the relation data, which otherwise, would be a cumbersome and difficult task. The fact that companies such as Microsoft and Oracle have recently added data mining extensions to their relational database management systems underscores their importance, and calls for a similar solution for RDF stores and SPARQL respectively.

The logic deduction capabilities of the Semantic Web allow to include additional inferred data in the data mining process that can be searched for patterns by learning algorithms. This should ultimately help to outperform statistical induction alone and open up new possibilities for the machine learning/data mining community.

1.2 Goals of the Thesis

This thesis presents a novel approach to apply existing data mining techniques on Semantic Web data. We introduce SPARQL-ML, an extension of SPARQL, which integrates data mining capabilities into the Semantic Web query language. By coupling the two, we hope to achieve more accurate predictions on Semantic Web data, and at the same time allow the Semantic Web research community to conduct data mining tasks with a familiar tool.

The goals of this thesis are, hence:

- to show that the inferred data can serve as useful additional information in the data mining process,
- to facilitate data mining tasks on Semantic Web data, such as classification or clustering, with the use of an integrated tool, and
- to provide users with a generic toolset, offering relational and propositional learning methods for the Semantic Web.

The thesis is structured as follows: the next section summarizes the most important related work, while Chapter 2 gives a short introduction into the concepts of Semantic Web data. In Chapter 3 we present different data mining techniques focusing on relational data mining. We also illustrate a possible combination of Semantic Web data and existing data mining methods. Chapter 4 shows the theoretical concepts of SPARQL-ML and in Chapter 5 we present our actual implementation of SPARQL-ML, which serves as a general framework for data mining tasks on Semantic Web data. We validate our approach in Chapter 6 with different experiments on synthetic and real-world datasets. We compare the results of classification algorithms applied to Semantic Web data with and without the support of inferencing. We also conduct an experiment to compare SPARQL-ML to an existing learning approach on Semantic Web data. Finally, we complete the thesis with the limitations, possible future work and the conclusions of our approach.

1.3 Related Work

Several recent studies focus on data mining from Semantic Web data or the enhancement of existing data mining tasks with inferred information from ontologies. We summarize these studies and work out the differences to our approach.

1.3.1 Data Mining from Semantic Web Data

Edwards *et al.* [Edwards et al., 2002] present an empirical investigation of learning from the Semantic Web, where they apply different machine learning methods to a typical user-profiling problem. The goal of their experiments is to learn a model which could then be used to recommend products to a user according to his profile. The authors test different datasets and compare the performance of learning from plain text format with learning from semantic meta-data. For the first experiment, they use traditional statistical machine learning methods. The results are not very promising, showing that the learning from semantically annotated data is not able to outperform the learning from plain text for that particular experiment. For the second experiment they apply the Progol Inductive Logic Programming (ILP) system, which is able to learn from supplied example instances and supporting background information. The results indicate some improvements: the algorithm is able to find a couple of reasonable rules for the classification task. They conclude that the Semantic Web markup available at that time cannot be expected to outperform conventional machine learning applied to plain text, with regards to the accuracy of the learned model. Our work extends this evaluation by looking at new statistical approaches appropriate for Semantic Web data and ontological support.

Closely related to our approach is the research on data mining from ontological data of Bloehdorn and Sure [Bloehdorn and Sure, 2007], in which they investigate on how to make current machine learning algorithms amenable to work on instances that are described by means of an ontological vocabulary. They present a framework for designing kernels that exploit the knowledge of underlying ontologies. For their implementation, they extend the Support Vector Machine (SVM) algorithm with adequate kernels on Semantic Web data. The framework is based on common notions of similarity. They introduce four different kernels, residing on different layers. The identity layer kernel solely considers the identity of two instances, while the class layer considers similarities of instances based on the classes they instantiate. The property layers on the other hand, compare the similarities of instances based on the data properties and/or object properties. To evaluate their framework, the authors present two experiments on different datasets. Experiment one tries to imitate the classification behavior of an ontology given a semantically weakened ontology. In a second experiment they predict the research group affiliation of persons and publications based on the SWRC ontology.[1] Inspired by this work, we conduct the same experiments using our SPARQL-ML approach and compare the results.

Chen *et al.* [Chen et al., 2003] use ontologies in a preprocessing step for attribute-value data to improve the results for association rule mining. They argue that real-world data is often too sparse to produce rules with reasonable support, for which reason they use the fact that ontologies with concept hierarchies can help to produce rules with improved support. They present their approach on a web marketing task, whose dataset consists of records of real personal data that contain demographic and expressed interests data. The data was gathered from portal sites like Yahoo and ICQ. When derivating association rules in the available data, the authors were not able to gain good rules because people tend to be very specific with their interests. This leads to the problem that only a few people in the data have a common interest and can be used to deduce a new rule. In order to circumvent this problem, Chen *et al.* created an ontology containing a concept hierarchy of interests. With the help of the ontology they raised the existing interest data to a higher concept level, where there are more people with the same interests. The experiment shows that the raised data was able to produce rules with a much larger support. In addition, they state that raising often creates rules which better represent the domain than rules created without raising. We follow a similar approach, and intend to enhance the data mining task with ontologies. We map the given ideas to the Semantic Web environment, where the data is already closely coupled with the concept of ontologies. The Semantic Web further allows the application of rich inferencing possibilities and does not solely focus on concept hierarchies. By providing an integrated tool for the Semantic Web, we furthermore skip the necessary step of preprocessing in order to raise the data.

Getoor and Licamele [Getoor and Licamele, 2005] introduce the importance of link mining for the Semantic Web and investigate the appropriate handling of correlations between entities. They state that the links among objects demonstrate certain patterns, which can be helpful for many data mining tasks and are usually hard to capture with traditional statistical models. We therefore apply relational learning algorithms, which take these patterns into account and improve the performance of the statistical models. Other studies discuss the discovery of association rules in RDF data by introducing new algorithms or operators [Jiang and Tan, 2006], [Anyanwu and Sheth, 2003].

Gilardoni *et al.* [Gilardoni et al., 2005] argue, that the main driver behind the need for machine

[1] `http://ontoware.org/projects/swrc/`

learning techniques for the Semantic Web is that the web is so huge. Therefore, support from tools that are able to work autonomously is needed. SPARQL-ML offers this automation support and helps to apply machine learning techniques on Semantic Web data to (i) find patterns in relational data, and (ii), to further annotate existing Semantic Web data.

1.3.2 Tool Support

Little work has been done so far on seamlessly integrating knowledge discovery capabilities into the Semantic Web. Kochut and Janik [Kochut and Janik, 2007] extended SPARQL with operators for semantic association discovery, allowing the search for semantic associations among entities in Semantic Web data. A semantic association is an undirected path that connects two entities in the knowledge base using named relationships, which represent its meaning. They extend SPARQL's triple patterns to include path patterns, which are triple patterns created with the use of a path variable in place of the property. During the pattern matching part, the path variable will be bound to the located paths between two resources. The result will be represented as a sequence of properties and the connecting resources, making out the path. Their evaluation shows the use of the path pattern for the search for semantic associations in data from biological sciences. We follow a similar approach for extending SPARQL with a prediction triple pattern. The triple should allow to predict the value of a variable by applying an existing mining model. The benefit of our approach is that we are able to use a multitude of different machine learning techniques to not only perform semantic association discovery, but also classification and clustering.

Hartmann [Hartmann, 2004] presents an approach to a knowledge discovery workbench that extends existing data mining methods with ontologies as background information. The workbench allows the interpretation and translation of Web documents into semantically enriched representations. For the data mining task it uses the ILP system Progol, which applies inverse entailment to generate only the most specific hypothesis. To further restrict the hypothesis space, the user has to manually define first-order expressions which define predicate and function symbols. Our approach differs from the one presented by Hartmann. By creating a principle framework for data mining from Semantic Web data we do not focus on one machine learning technique but allow the use of different propositional and relational learning algorithms. For our thesis we will focus on the techniques provided by statistical relational learning that avoid the task of having to rewrite Semantic Web datasets into logic programming formalisms.

1.3.3 Web Service Classification

Hess and Kushmerick [Hess and Kushmerick, 2004] present a machine learning approach to semi-automatically classify web services. The goal of their experiments was to create an application, which determines the category of a web service description and recommends it to the user for further annotation. The authors state that the user would save a considerable amount of work if he or she only had to choose between a few predicted categories. They treated the determination of a web service's category as a text classification problem and applied traditional data mining algorithms, such as Naive Bayes and SVM. The dataset used for the experiments consists of WSDL web services.[2] We present a similar experiment on OWL-S web service descriptions.[3] Our clas-

[2] http://www.w3.org/TR/wsdl

[3] http://www.daml.org/services/owl-s/1.1/

sification of web services makes use of relational learning algorithms and additional inferred knowledge provided by ontologies. We propose that, due to the given structure of Semantic Web service descriptions, relational learning algorithms are able to create more accurate models than propositional learning methods.

Sabou [Sabou, 2005] states that the Semantic Web can facilitate the discovery and integration of web services. The addition of ontologies, containing knowledge in the domain of the service, such as the type of an input or output parameter, offers new background information that can be exploited by machine learning algorithms. We evaluate this assumption in our work by comparing the results of machine learning algorithms applied on Semantic Web data with and without ontology support. Furthermore we focus on the input and output parameters of Semantic Web service descriptions to correctly classify the web services.

2

Characteristics of Semantic Web Data

The Semantic Web enhances the traditional web by adding a semantic layer on top of the well-known web data formats to make the web machine readable. In this chapter we introduce the basic principles and characteristics of Semantic Web data, which will be necessary for the understanding of the remainder of this thesis.

2.1 Structure

As the corner stone for describing data in such a manner, the Resource Description Framework (RDF) has been created. The RDF Specification provides the following definition:[1] "RDF is based on the idea of identifying things using Web identifiers (called Uniform Resource Identifiers, or URIs), and describing resources in terms of simple properties and property values". RDF can be described by its graph data model which states that the underlying structure of an RDF expression is a collection of triples, each consisting of a subject, a predicate and an object. A triple can be illustrated as a node-arc-node link as shown in Figure 2.1.

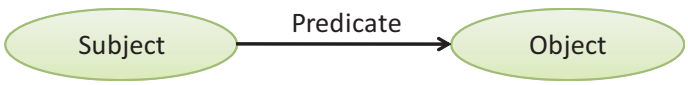


Figure 2.1: RDF triple composed of subject, predicate, and object

The link represents a relation between the subject and the object, while the direction of the predicate always points toward the object. The object can be a resource or a literal, whereas the subject must not be a literal. If the object is a resource, it can itself be the subject of another triple pointing to another object. A set of such triples is called an RDF graph.

[1] http://www.w3.org/TR/rdf-primer/

2.2 Semantics

The OWL Web Ontology Language [Mcguinness and van Harmelen, 2004] allows an even greater machine interpretability of the web by providing additional vocabulary and formal semantics to make the data more expressive. It serves as a standard language to define the terms in vocabularies and the relationships between those terms. Opposed to databases, ontologies serve as conceptual structures to describe the entire application domain, instead of just describing one specific application. The basic elements of an ontology are as follows:

- **Classes** serve as an abstraction mechanism for grouping resources with similar characteristics. Every class is associated with a set of individuals, called the class extension.
- **Individuals** represent the instances of a class.
- **Object properties** link individuals to individuals with restrictions for the domain and the range.
- **Datatype properties** link individuals to data values with restrictions for the domain and the range. The datatypes defined by OWL rely on the XML Schema datatypes listed by Biron and Malhotra [Biron and Malhotra, 2004].

Through the application of a reasoner on Semantic Web data, we can infer additional triples from a given ontology. Hence, inferencing helps implicit knowledge to become explicit. This thesis argues that the implicit knowledge can serve as useful additional information in the data mining process. We will explain the use of ontologies in the data mining tasks in Chapter 3.

2.3 Querying

The RDF Data Access Working Group created a W3C recommendation for the querying of the Semantic Web with the RDF query language SPARQL [Prud'hommeaux and Seaborne, 2007].[2] It consists of the syntax and semantics for the querying against RDF graphs. Therefore, the core of the query language is based on matching graph patterns. The graph patterns contain triple patterns which are similar to RDF triples, but with the option of replacing an RDF term in the subject, predicate or object position with a query variable. The variables inside a triple pattern are identified through the '?' prefix. SPARQL also allows the use of conjunctions, disjunctions, and optional patterns. Listing 2.1 gives a simple example of the syntax of a SPARQL query.

A query mainly consists of the following parts: the *prologue* (line 1), which contains the definition of namespace prefix bindings. This allows a user to write the prefix inside a query instead of rewriting the whole URI again. The desired *output* of a SPARQL query is defined through the query type, which is a `SELECT` query in our example (line 3). The main part is the *basic graph pattern (BGP)* (lines 4-7), which holds all the triple patterns to be matched to the underlying RDF graph. Finally a SPARQL query may include *solution modifiers* (line 8), which modify the output of the pattern matching with classical operators such as distinct, order, limit, and offset.

When querying the example dataset about different persons in Listing 2.2, the example query in Listing 2.1 looks for the resources $?x$ and their given name $?givenName$. The only constraint

[2] `http://www.w3.org/2001/sw/DataAccess/`

```
1 PREFIX vcard: <http://www.w3.org/2001/vcard-rdf/3.0#>
2
3 SELECT ?x ?givenName
4 WHERE {
5  ?x vcard:Family "Smith" .
6  ?x vcard:Given  ?givenName .
7 }
8 LIMIT 1
```

Listing 2.1: SPARQL SELECT query with a LIMIT solution modifier.

```
1 @prefix vcard: <http://www.w3.org/2001/vcard-rdf/3.0#> .
2
3 <http://example.org/person#John>
4         vcard:Family "Myers" ;
5         vcard:Given "John" ;
6         vcard:Prefix "Dr" .
7
8 <http://example.org/person#Robert>
9         vcard:Family "Smith" ;
10        vcard:Given "Robert" ;
11        vcard:Prefix "Prof" .
12
13 <http://example.org/person#Claudia>
14        vcard:Family "Smith" ;
15
16 <http://example.org/person#Sarah>
17        vcard:Family "Richards" ;
18        vcard:Given "Sarah" ;
```

Listing 2.2: Example RDF dataset.

in the *pattern matching part* is that the resources need to have the family name 'Smith' and an existing given name. The final output of query 2.1 is shown in Table 2.1. The person with the family name 'Smith' and an existing given name is the resource *http://example.org/person#Robert*. Although the result is limited to one record, the solution is not changed since the data has only one resource that matches the BGP.

x	*givenName*
<http://example.org/person#Robert>	Robert

Table 2.1: Output of the SPARQL `SELECT` query in Listing 2.1.

SPARQL also introduces an `OPTIONAL` keyword to be used in the BGP, which basically allows to define non mandatory triples. If the graph matching algorithm finds solutions for the variables inside the optional clause it adds the additional bindings to the solution, otherwise it does not return a value for the optional variables. The query in Listing 2.3 again looks for the resources

in the data in Listing 2.2 with the familiy name 'Smith', but we now enclosed the triple with the given name property in an `OPTIONAL` clause. The results of this query are shown in Table 2.2. The query now returns all resources $?x$ and their given names $?givenName$ that have the family name 'Smith' and an optional given name. The final output consists of two records, of which the resource *http://example.org/person#Claudia* does not have a given name.

```
1 PREFIX vcard: <http://www.w3.org/2001/vcard-rdf/3.0#>
2
3 SELECT ?x ?givenName
4 WHERE {
5  ?x vcard:Family "Smith" .
6  OPTIONAL { ?x vcard:Given  ?givenName }
7 }
8 LIMIT 1
```

Listing 2.3: SPARQL SELECT query with an OPTIONAL clause.

x	*givenName*
<http://example.org/person#Robert>	Robert
<http://example.org/person#Claudia>	

Table 2.2: Output of the SPARQL `SELECT` query in Listing 2.3.

3

Knowledge Discovery for the Semantic Web

To introduce our approach of Knowledge Discovery for the Semantic Web (KDSW), we first have to succinctly summarize the main terms used in the related area of Knowledge Discovery in Databases (KDD). On this basis we can then elaborate on the techniques and methods used in KDD to present their possible integration into the Semantic Web.

3.1 Definitions

To avoid misunderstandings, we will first define the most important terms used in conjunction with KDD by summarizing the definitions given by Fayyad *et al.* [Fayyad et al., 1996].

- **Knowledge Discovery in Databases** is the non-trivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data.
- **Data Mining** is solely a step in the KDD process concerned with applying computational techniques to actually find patterns in the data.

The mentioned patterns consist of hidden predictive information, which helps us to solve different problems, such as the classification or clustering of data. The traditional data mining approach expects as input a table of independent instances. This approach is also called propositional data mining. The Semantic Web data on the other hand is highly relational, with instances depending on other instances. This situation demands for relational data mining techniques, which take the additional dependencies of the heterogeneous and complex structure of interlinked Semantic Web data into account. Due to the nature of Semantic Web data, this thesis focuses on relational learning techniques, although our approach also supports propositional learning techniques.

The KDD process also includes steps for the data preparation and the evaluation of the discovered patterns during the data mining process. The development and design of algorithms for data mining can be summarized with the term machine learning.

3.2 Propositional Data Mining

In propositional data mining, the input is considered to be a set of instances, which are independent individuals of the concept to be learned. These instances are characterized by the values of attributes, which measure different aspects of the individuals. Most of the data mining tasks today follow this approach, for which there are numerous algorithms available.

The available algorithms create different data mining models, which have different knowledge representations. We introduce the three main distinctions as proposed by Liu [Liu, 2007].

3.2.1 Supervised Learning

Probably the most often applied learning method in machine learning is supervised learning, which is also called classification or inductive learning and involves the process of learning by example. The algorithm tries to induce a general rule from a set of observed instances and then assigns to an unclassified instance, the name of a class to which it belongs. The mining model learned through supervised learning is called classification model, predictive model or classifier. The instances in the supervised learning task have a set of attributes $A = \{A_1, A_2, ..., A_{|A|}\}$, where $|A|$ denotes the number of attributes. Additionally, the instances have a target attribute C, which is called the class attribute. In the training examples the class attribute is known, for which reason the learning is called supervised.

Supervised learning methods can also be applied to Semantic Web data. As described before, the input instances for a propositional learning algorithm have to be independent and homogeneous. This is mostly not the case in relational data (*e.g.* Semantic Web data), which is why the concept of supervised learning has also been extended to the relational setting. Due to the wide spread use and applicability to relational data, this thesis focuses on classification models for Semantic Web data.

3.2.2 Unsupervised Learning

In unsupervised learning, the instances used for learning a mining model have no pre-defined classes. The algorithm has to find the hidden structures and regularities in the data by itself. For this purpose, the instances are organized into similarity groups, also called clusters. Instances in the same cluster are similar to each other, while instances in different clusters are different from each other. Clustering methods can also be applied to Semantic Web data, although similarity measures have to include additional similarity concepts because of the heterogeneous structure of the instances.

3.2.3 Association Rules and Sequential Patterns

The concept of association rule mining is very prominent in the data mining research due to its applicability in real world situations. The classic application of association rule mining is the market basket data analysis, which tries to discover the buying patterns of customers in a supermarket. Basically, an association rule represents a co-occurrence relationship between two items. In the supermarket example, the co-occurrence of cornflakes and milk in a supermarket

basket could be treated as an association rule, which states that people who buy cornflakes also buy milk. From this information we can conclude that to maximize our profit, we can for example arrange the cornflakes and the milk in the same shelf. Sequential patterns expand the pattern detection to find co-occurrences of items in some sequence.

3.3 Relational Data Mining

Most of the time, the necessary single table representation for propositional data mining is not enough. For some tasks, complex relations between different objects have to be taken into account. In order to allow a different input for the data mining process, we have to introduce the concepts of relational data mining. Getoor and Taskar [Getoor and Taskar, 2007] state that it is necessary to include the rich logical structure of the underlying data for solving more general and complex problems. Therefore, the relational approach looks for patterns that involve multiple relations, instead of concentrating on a single table representation of the data.

Relational data basically violates two assumptions made by traditional data mining techniques as stated by Neville *et al.* [Neville et al., 2003b]:

1. The instances in relational data are not recorded in a homogeneous structure. They consist of sets of heterogeneous records.

2. The instances in relational data are not independent and identically distributed. They have dependencies through relations and through chaining multiple relations together.

Working with relational data in the data mining process requires the ability to handle structurally heterogeneous and dependent data instances, for which new algorithms need to be developed. Most techniques for relational data that are used today are based on inductive logic programming (ILP, [Dzeroski, 2003]), which is concerned with finding deterministic patterns expressed as logic programs. ILP is situated at the crossing of the research fields of logic programming and machine learning. The patterns found by ILP methods are expressed in first-order logic. Since the process of defining logic programs is a complex task, relational data mining research has begun to incorporate probabilistic representations in their algorithms. This new field, joining relational data mining and probabilistic learning, is called Statistical Relational Learning (SRL). Neville *et al.* [Neville et al., 2003c] have listed a number of advantages of SRL, stating that probabilistic relational models are better than deterministic models in most real-world data classification tasks.

3.3.1 Statistical Relational Learning

Due to the similar structure of probabilistic relational models and Semantic Web data, we argue that relational learning algorithms based on probabilistic models are an ideal candidate for the use with Semantic Web data. The structure of Semantic Web data (see Chapter 2) consists of heterogeneous data, interconnected with different directed links. We can, therefore, construct a graphical model of probabilistic relations of Semantic Web data. Our model is heavily influenced by the representation of Probabilistic Relational Models (PRM, [Getoor et al., 1999]) and Relational Dependency Networks (RDN, [Neville and Jensen, 2004]).

To use the standard graph syntax of the Semantic Web, we use circles to represent resources as objects O_n, rectangles to represent literals as attributes A_n, solid lines to represent properties as links, and dashed lines to represent probabilistic dependencies. Every object has a number of associated datatype properties linked to attributes $\{A_1, ..., A_n\}$, and a number of object properties linking to or from other objects $\{O_1, ..., O_n\}$. Consider for example Figure 3.1, where the objects Company, Employee, Project, and their relations are illustrated. We define the attribute $Success$ of the project object and, therefore, the datatype property $isSucess$ to be our class label for a classification task.

The attributes of an object can depend probabilistically on other attributes of the same object, as well as on attributes of other related objects. The dependencies between the attribute $Success$ and other attributes are represented in the model with dashed lines. We omitted other dependencies to keep the graph as simple as possible. The success of a project may depend on the budget ($Budget$) allocated for the project, on the work experience ($Experience$) of employees working at the project, or on the popularity ($Popularity$) of the company financing the project. Each attribute is associated with a probability distribution conditioned on other attributes. The dependent attributes of attribute A_n^j are therefore either associated with the same object O_j, or with another object O_k linked to O_j through an object property or through a chain of object properties. If the relation between O_j and O_k is one-to-many, the dependent attribute consists of a set of attribute values. For this purpose, the model uses aggregation functions, either to create a single value out of a set of values, or to combine a set of probability distributions into a single distribution. In our case, this situation can occur for the object property between the objects Project and Employee. The attribute $Success$ can depend on more than one employee, and hence, on a set of attribute values of the attribute $Experience$.

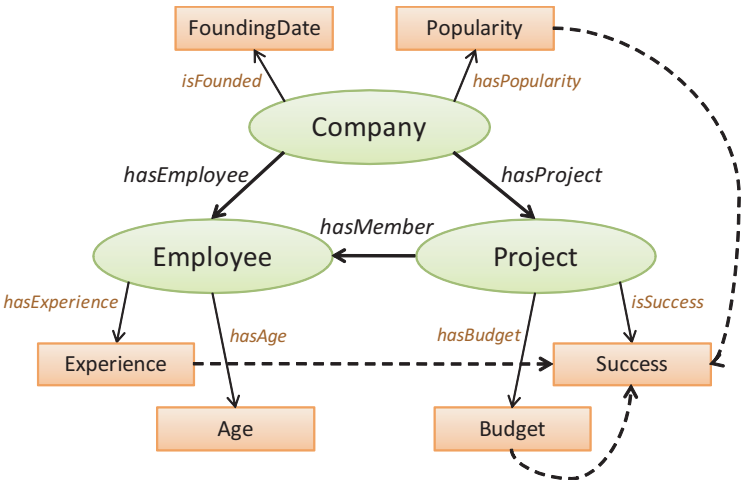


Figure 3.1: Probabilistic relational graph showing the dependencies between attributes of different objects.

In order to apply data mining tasks on relational models, new machine learning algorithms have been developed. The resulting algorithms are able to create the necessary conditional probability distributions over possible class label values.

For our evaluation in Chapter 6, we particularly applied two well-known algorithms: the Relational Bayes Classifier (RBC, [Neville et al., 2003b]) and the Relational Probability Tree (RPT,

[Neville et al., 2003a]). Following, we will summarize the basic principles of these algorithms.

Relational Bayes Classifier (RBC)

An RBC is a modification of the traditional Simple Bayesian Classifier (SBC) for relational data. The SBC assumes that attributes are conditionally independent of its class C. The RBC applies this independence assumption to relational data. Before being able to estimate probabilities, the RBC decomposes (flattens) structured examples down to the attribute level. Heterogeneous subgraphs are being transformed to homogeneous sets of attributes. Figure 3.2 shows an example instance (subgraph) to predict the success of a business project in a relational dataset. The available attributes for the prediction include the age and the experience of the employees. Due to the possibility that a project can have multiple employees, the attributes contain a multiset of values for each subgraph. The same data can be decomposed to the representation shown in Table 3.1. Each row makes out one subgraph, while each column represents an attribute. The cells contain the sets of values for each subgraph.

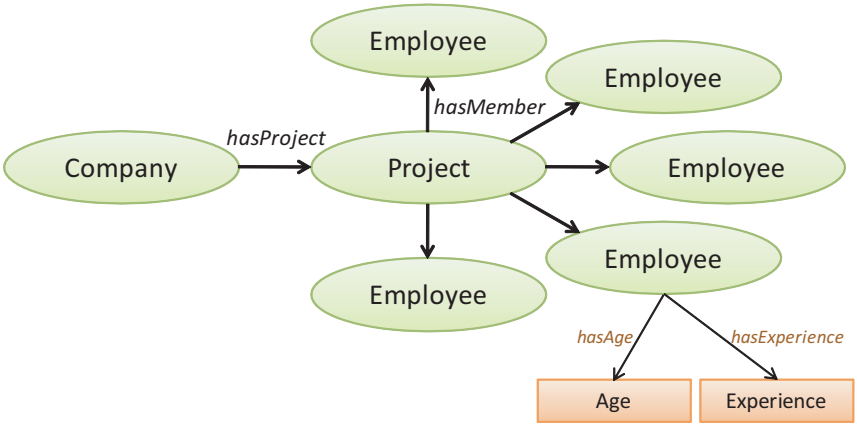


Figure 3.2: Relational data represented as a subgraph [Neville et al., 2003b].

Success	Employee Age	Employee Experience	...
YES	23,25,37,32,41	2,4,3,1,12	...
NO	18,25	1,7	...
YES	17,26,44	1,3,17	...
...	...	...	...

Table 3.1: Relational data decomposed by attribute.

The RBC then follows the technique of the SBC, which states that the probability of a class C given an example can be computed as the product of probabilities of the example's attributes $A_i, ..., A_n$ given the class (see Equation 3.1).

$$P(C = + \mid A_1, ..., A_n) = \alpha P(C = +) \prod_{i=1}^{n} P(A_i \mid C = +) \tag{3.1}$$

In order to estimate the probabilities of each attribute given the class, the RBC uses different estimation techniques. The *average value* estimator averages the multisets of values after the decomposition to the attribute level. Continuous values will, therefore, be replaced with their average and discrete values will be replaced with the modal value. For example $P(+\,|\,E) = \alpha P(Average_{Age} = 29\,|\,+)P(+)$ computes the probability of a project E being successful with an average age of 29 for its employees.

The *independent value* estimator considers each value of each set to be an independent instance. This increases the amount of instances for estimation to the number of linked objects with the specified attribute. After the probability of each value has been computed, they are multiplied into an overall probability, *i.e.* $P(+\,|\,E) = \alpha P(17\,|\,+)P(26\,|\,+)P(44\,|\,+)P(+)$. The outcome will then be used for the final probability computation with the help of Equation 3.1. Neville *et al.* evaluate the different estimators and identify the pros and cons of each approach in [Neville et al., 2003b].

Relational Probability Tree (RPT)

The RPT extends standard probability estimation trees to a relational setting in which data instances are heterogeneous and interdependent [Neville et al., 2003a]. The input for the RPT algorithm consists of a collection of subgraphs, of which each subgraph contains a single target object to be classified. Similar to the RBC, the RPT looks beyond the attributes of the item for which a prediction should be made. It also considers objects of the relational neighborhood to estimate the conditional probability distribution. After having decomposed the structure down to multisets of attributes as illustrated in Table 3.1, the RPT searches over a space of binary relational features to split the data. The features are created by mapping sets of values into a single value with the help of aggregation functions. The RPT offers the following set of aggregation functions: MODE, AVERAGE, COUNT, PROPORTION, MINIMUM, MAXIMUM, EXISTS and DEGREE. Almost all of them are common aggregation functions for relational data, except of the DEGREE function. It allows to use the structure of relational data as a feature in the probability tree by counting the number of specific nodes in the subgraph. An example RPT is shown in Figure 3.3.

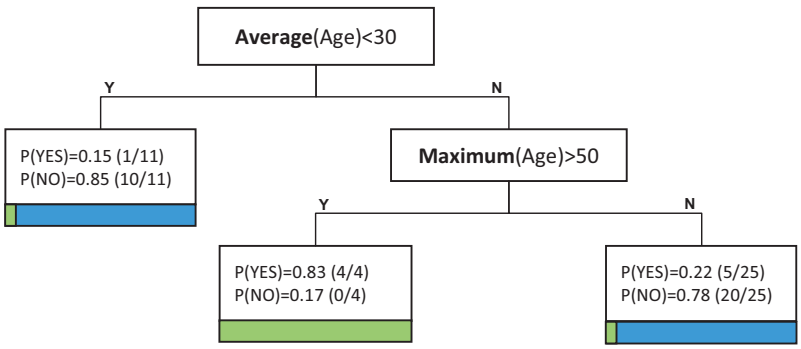


Figure 3.3: Example RPT with two binary features for the attribute Age.

In order to split the data according to a feature, the RPT looks for a possible threshold that could be used in conjunction with an aggregated attribute. A simple feature for example states that the average age of the employees in a project has to be smaller than 30 ($Average(Age) < 30$).

If a new project with an average age of 28 arrives at that specific feature in the decision tree, it computes the feature ($28 < 30$) and according to the result moves further down the yes or no branch. For this example project, it would move further down the yes branch since 28 is indeed smaller than 30.

In order to decide which feature to use at which point in the decision tree, the algorithm calculates the score for each feature. The score measures the correlation with the help of a chi-square test on the class counts after splitting the training data according to a feature. Additionally the algorithm calculates the p-value associated with the feature score and drops features with a non-significant p-value. Among the remaining features, the feature with the best score is chosen to be included in the probability tree. If the algorithm cannot find any feature with a significant p-value, the tree growing stops and no further splits are applied. Finally, the algorithm calculates the class distribution of the training examples at each leaf with the application of a laplace correction [Domingos and Provost, 2000]. For example, the class distribution of the leftmost leaf is 1 positive class and 10 negative classes. The probability of a class at a leaf can be calculated with the frequency-based estimate $\frac{p}{N}$, where we have p examples of the class in question and N total examples at a leaf. The probability would thus be $\frac{1}{11} = 0.09$ for the positive and $\frac{10}{11} = 0.91$ for the negative class. The laplace correction improves the probability estimates by introducing a prior probability of $\frac{1}{C}$ for each class, where C is the total number of classes. The final estimate would therefore be $\frac{p+1}{N+C}$, which leads to a probability of $\frac{1+1}{11+2} = 0.15$ for the positive and $\frac{10+1}{11+2} = 0.85$ for the negative class. The laplace correction has become standard procedure in machine learning for the smoothing of probability estimates from small samples.

3.4 Semantic Web Mining

After having presented the characteristics of Semantic Web data and the concepts of relational models, we will now present our approach on merging the two similar structures into a single model for relational data mining on Semantic Web data.

The Semantic Web provides additional, sometimes crucial, inferred knowledge, which does not exist in common relational data. With the use of inferencing we can go one step further and improve the performance of learning from relational data by including implicit information given by additional relations of individuals to or from classes and/or properties. In the following, we illustrate example situations in which ontologies can be a benefit for the data mining task.

3.4.1 Implicit Class Knowledge

Ontologies allow the definition of classes, as well as whole class hierarchies. Consider for example the business ontology shown in Figure 3.4. The ontology further refines the Person class into subclasses, such as Manager or Non-Manager. On a next level we can refine these new classes into for example production, sales, or marketing employees. This constitutes a class hierarchy with a strict IS-A relationship between the classes.

According to the representation of subgraphs in the previous Section, we model our data as a subgraph shown in Figure 3.2. The only difference now is that we substitute the employee objects with individuals that instantiate our classes in the ontology and get a new subgraph representation such as the one shown in Figure 3.5. For reasons of readability we omitted the company

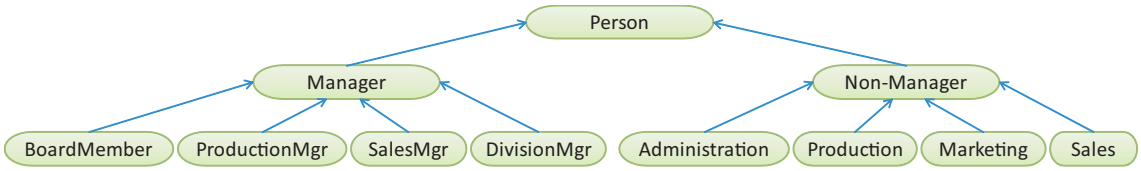


Figure 3.4: Example business ontology.

object and the attributes.

The new subgraph now not only contains explicit relations between employees and projects but also implicit relations between instantiated classes and their respective superclasses. These additional relations determine whether an employee is a manager or not, and of course, whether or not an employee is a person, which is the case for all employees since it is the root class of all existing classes in the ontology. Our previous classification task, which has the goal to predict the success of a business project, can now use these relations in the learning phase for a more accurate prediction. We defined that most projects are successful, if more than three employees working at a project are managers. Therefore, the best predictor is hidden in the ontology as implicit information. With the help of inferencing it is possible to use this information in the data mining task.

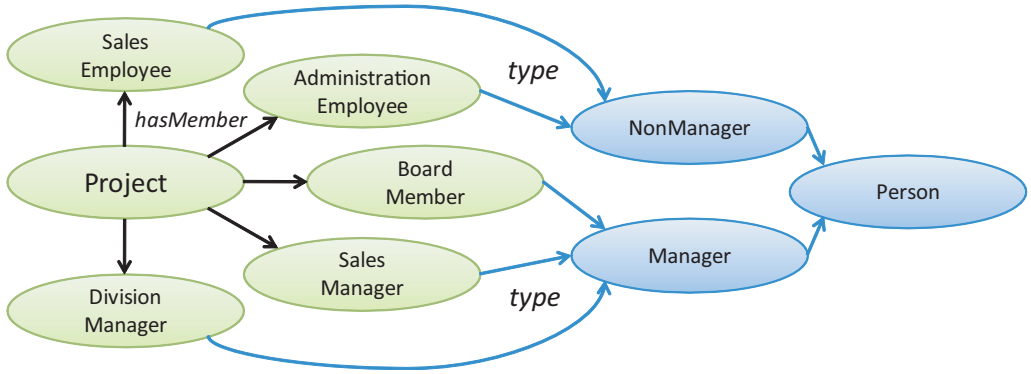


Figure 3.5: Relational data represented as a subgraph with inferred information about classes.

```
1 ?project ex:isSuccess ?success .
2 ?project ex:hasTeam ?employee .
3 ?employee rdf:type ?class .
```

Listing 3.1: SPARQL graph patterns with rdf:type predicate.

The SPARQL basic graph pattern (BGP) shown in Listing 3.1 matches all projects and its associated employees. The triple consisting of an $?employee$ subject, an *rdf:type* property, and the $?class$ object looks for the type of the employee individuals. Without inferencing, this triple pattern would only match the direct types of an individual. The same BGP applied on an inferred model also matches the super classes of the direct types for the $?class$ variable.

We can, therefore, extend the graphical model of probabilistic relationships presented in Section 3.3.1 with a type object. Every object O_n can be associated with one or more types. Since each

type only exists once in every subgraph but can be associated with more than one object at the same time as shown in Figure 3.5, we have to consider the links as a measure for type occurrence in a subgraph. The subgraph in Figure 3.5 for example contains three links to the type `Manager` and two links to the type `NonManager`. The relational learning algorithms can calculate aggregations on this new feature and include it in the data mining task. In the case of an RBC, this adds a new attribute to be used to estimate a conditional probability given the class. The RPT can compute features on this information with the help of the available aggregation functions introduced in Section 3.3.1.

3.4.2 Implicit Property Knowledge

Ontologies also allow the definition of object and datatype properties for classes. These definitions state which links and what literals a class is allowed to have. Listing 3.2 presents an example definition of an object property *hasBenefit*. It has the range *Stocks* or *Bonus*, indicating what kind of benefits the classes in the domain can have besides their salary. Additionally, a minimum cardinality of one is set, which means that every class in the domain must at least have one benefit.

```
1 <owl:ObjectProperty rdf:ID="hasBenefit">
2
3  <rdfs:domain>
4   <owl:Class>
5    <owl:unionOf rdf:parseType="Collection">
6     <owl:Class rdf:about="#SalesManager"/>
7     <owl:Class rdf:about="#DivisionManager"/>
8     <owl:Class rdf:about="#BoardMember"/>
9    </owl:unionOf>
10  </owl:Class>
11 </rdfs:domain>
12
13 <rdfs:range>
14  <owl:Class>
15   <owl:unionOf rdf:parseType="Collection">
16    <owl:Class rdf:about="#Stocks"/>
17    <owl:Class rdf:about="#Bonus"/>
18   </owl:unionOf>
19  </owl:Class>
20 </rdfs:range>
21
22 <owl:minCardinality>1</owl:minCardinality>
23
24 </owl:ObjectProperty>
```

Listing 3.2: Object property definition.

In real-world datasets we often have the problem of sparse data, which means that we have attributes with lots of null or zero values. It is difficult to achieve good results for a learning algorithm on such a dataset due to insufficient data and potential incorrect treatment of missing values. In the following we illustrate an example situation, where the enhancement of the data mining task with property definitions can help to overcome the problem of sparse data.

Our example ontology contains the object property definition in Listing 3.2 and the definition of 8 classes, all being subclasses of the `Person` class. The class hierarchy of our ontology is shown in Figure 3.6.

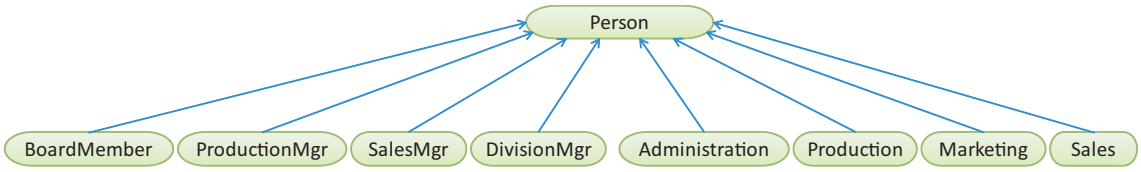


Figure 3.6: Example business ontology.

We once again want to predict whether a business project will be successful or not. Unlike before, we now define a project to be successful if the project has more than three team members with a *hasBenefit* property. In order to correctly simulate the situation of sparse data, our virtual dataset contains only a few individuals with an actual *hasBenefit* property. The little explicit information is therefore not enough to train a good classifier. With inferencing, it is possible to infer additional triples, indicating, whether or not a team member has a benefit. By making the implicit link to the property explicit, we eliminate the problem of sparse data and provide the learning algorithm with a good predictor for the classification task. Figure 3.7 illustrates an abstract subgraph for our classification task, showing the implicit links between the types and the property definition in the ontology.

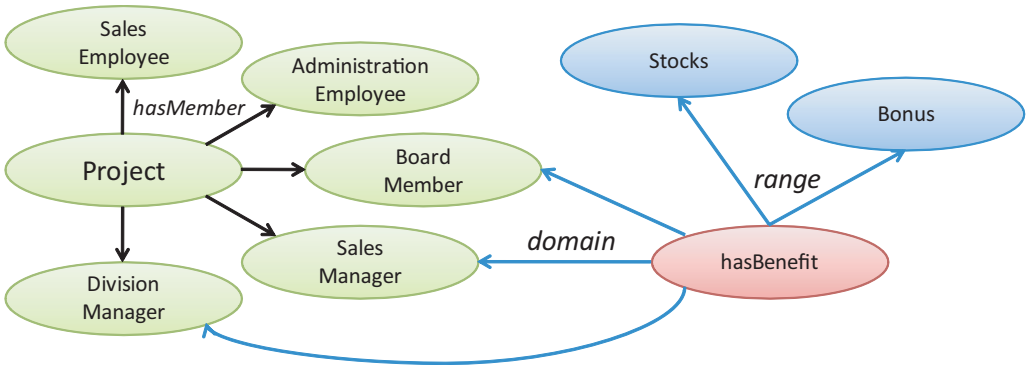


Figure 3.7: Relational data represented as a subgraph with inferred information about properties.

Similar situations, where property definitions of an ontology can provide the data mining task with inferred knowledge, can also be shown with datatype properties.

4

SPARQL-ML

SPARQL-ML (SPARQL Machine Learning) is an extension of SPARQL, which supports data mining tasks for knowledge discovery in the Semantic Web. Our extension adds new syntax elements and semantics to the official SPARQL grammar described by Prud'hommeaux and Seaborne [Prud'hommeaux and Seaborne, 2007].

SPARQL-ML facilitates the following two tasks on any Semantic Web dataset: (1) train/learn/induce a model based on training data using the new `CREATE MINING MODEL` statement (Section 4.1); and (2), apply a model to make predictions using the `PREDICT` statement (Section 4.3). The model created in the `CREATE MINING MODEL` step follows the definitions in our *SPARQL Mining Ontology* (SMO) also presented in Section 4.2.

4.1 Learning a Model

With the use of our `CREATE MINING MODEL` statement, it is possible to induce a classifier (model) on any Semantic Web training data. The chosen syntax was inspired by the Microsoft Data Mining Extension (DMX) that is an extension of SQL to create and work with data mining models in Microsoft SQL Server 2005 Analysis Services (SSAS).[1]

The extended SPARQL grammar is tabulated in Table 4.1 and Listing 4.1 shows a particular example query for our business project success prediction task in Section 6.2. Our approach adds the *CreateQuery* symbol to the official SPARQL grammar rule of *Query*. The structure of *CreateQuery* resembles the one of *SelectQuery*, but has complete different semantics: the *CreateQuery* expands to Rule [1.1] adding the new keywords `CREATE MINING MODEL` to the grammar followed by a *SourceSelector* to define the name of the trained model. In the body of *CreateQuery*, the variables to train the model are listed. Each variable is specified with its content type (Table 4.2), which enables different processing during the learning process: variables of the type `CONTINUOUS` allow the use of mathematical operations like *average*, *minimum*, and *maximum*. `DISCRETE` variables can use mathematical operations like *mode* and *exists*. The `RESOURCE` type holds an RDF resource (IRI or blank node). The content type is different from the datatype, because we only differ between three content types, while Semantic Web data is allowed to have more different datatypes as listed in Table 4.2.

[1] `http://technet.microsoft.com/en-us/library/ms132058.aspx`

In order to specify which variable should be learned by the model, we introduced two keywords: `PREDICT` tells the learning algorithm that this feature should be predicted. Additionally, one variable of the content type `RESOURCE` has to be specified with the `TARGET` keyword to denote the resource for which a feature should be predicted.

After the usual *DatasetClause*, *WhereClause*, and *SolutionModifiers*, we introduced a new *UsingClause*. The *UsingClause* expands to Rule [1.2] that adds the new keyword `USING`, followed by a *SourceSelector* and an optional *BrackettedExpression* to define the name and additional parameters for the learning algorithm.

[1]	*Query*	::=	Prologue(SelectQuery | ConstructQuery | DescribeQuery | AskQuery | CreateQuery)
[1.1]	*CreateQuery*	::=	'CREATE MINING MODEL' SourceSelector '{' Var 'RESOURCE' 'TARGET' (Var ('RESOURCE' | 'DISCRETE' | 'CONTINUOUS') 'PREDICT'?)+ '}' DatasetClause* WhereClause SolutionModifier UsingClause
[1.2]	*UsingClause*	::=	'USING' SourceSelector BrackettedExpression

Table 4.1: Extended SPARQL grammar for the `CREATE MINING MODEL` statement.

```
1  PREFIX rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2  PREFIX ex:      <http://www.examplecompany.org/>
3
4  CREATE MINING MODEL <http://www.example.org/models/projectSuccess> {
5   ?project RESOURCE TARGET
6   ?success DISCRETE PREDICT {'YES','NO'}
7   ?member RESOURCE
8   ?class RESOURCE
9  }
10 WHERE {
11  ?project ex:isSuccess ?success .
12  ?project ex:hasTeam ?member .
13  ?member rdf:type ?class .
14 }
15 USING <http://www.kdl.cs.umass.edu/proximity/rpt>
```

Listing 4.1: SPARQL-ML `CREATE MINING MODEL` query.

Content type	Description	Example datatypes
RESOURCE	Content is a resource (IRI or blank node)	RDF resource
DISCRETE	Content is a literal with a discrete value	string, date
CONTINUOUS	Content is a literal with a continuous value	integer, double, float

Table 4.2: SPARQL-ML content types and supported datatypes.

Semantics of `CREATE MINING MODEL` Queries

According to Pérez *et al.* [Perez et al., 2006], a SPARQL query consists of three parts: the *pattern matching part*, the *solution modifiers*, and the *output*. In that sense, the semantics of the CREATE MINING MODEL queries is the construction of new triples describing the metadata of the trained model (*i.e.*, a new output type). The metadata follows the definitions of our SPARQL Mining Ontology, which is introduced in the next Section. The ontology enables to permanently save the parameters of a learned model that are needed by the `PREDICT` query (see Section 4.3).

4.2 SPARQL Mining Ontology (SMO)

The SPARQL Mining Ontology (SMO) defines the necessary concepts to permanently save the metadata of a learned model. By introducing these common definitions, we are able to seamlessly integrate additional machine learning techniques into SPARQL-ML. Furthermore, it is possible to query the metadata of different mining models with a standard SPARQL query. At runtime, the `PREDICT` query can access the stored information of a specified model to correctly map the input data to the feature definitions of the model. The SMO is illustrated in Figure 4.1 while Listing 4.2 shows an extract of an example metadata entry of a mining model.

```
1  @prefix smo:     <http://www.ifi.uzh.ch/sparql/mining/> .
2  @prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
3
4  <http://www.example.org/projectSuccess>
5     smo:hasModelFile <http://www.example.org/projectSuccess/model_RPT.xml> ;
6     smo:hasFeature <http://www.example.org/projectSuccess#project> ;
7     smo:hasFeature <http://www.example.org/projectSuccess#success> ;
8     smo:usesAlgorithm <http://www.kdl.cs.umass.edu/proximity/rpt> ;
9     smo:hasModelName "projectSuccess" ;
10    a       smo:Model .
11
12 <http://www.example.org/projectSuccess#success>
13    smo:hasVarName "success" ;
14    smo:isPredict "1" ;
15    smo:hasFeatureType "DISCRETE" ;
16    smo:hasLink <http://www.example.org/projectSuccess/link/isSuccess> ;
17    smo:hasNominalValues _:b1 ;
18    a       smo:Feature .
19
20 <http://www.example.org/projectSuccess/link/isSuccess>
21    smo:linkName "isSuccess" ;
22    smo:linkFrom <http://www.example.org/projectSuccess#project> ;
23    a       smo:Link .
24
25 _:b1  rdf:li "NO" ;
26    rdf:li "YES" ;
27    a       rdf:Bag .
```

Listing 4.2: SPARQL mining ontology example metadata.

Lines 4–10 show the constructed triples of a model with the name *projectSuccess*. The model uses the algorithm described by the resource *http://kdl.cs.umass.edu/proximity/rpt*, which defines the necessary parameters for a Relational Probability Tree. The model also contains a property *hasModelFile* pointing to the actual file of the mining model. The model file is used in the mining module that will be introduced in the next Chapter. The metadata also contains the specified features of the model through the *hasFeature* property. The model in Listing 4.2 has the features *project* and *success*. On lines 12–27 the feature *success* is further described through its variable name (*hasVarName*), the content type (*hasFeatureType*), whether the feature is to be predicted or not (*isPredict*), the given nominal values (*hasNominalValues*), and the links (*hasLink*) to other features.

The metadata of the algorithms and mining modules is also stored as RDF data and follows the SMO (see Appendix B). Table 4.3 describes the classes representing parts of the mining model and Table 4.4 lists all datatype properties and their domain, revealing for which class this property is applicable.

Name	Description
Model	A data mining model.
ModelFile	A file resource that contains the actual mining model.
Feature	Represents a feature of a mining model.
Link	Represents a link between two features of a mining model.
Algorithm	A data mining algorithm.
Param	A parameter of a data mining algorithm.
MiningApp	A mining application.

Table 4.3: Classes of the SPARQL mining ontology.

Name	Type	Domain	Description
hasModelName	string	Model	The name of the mining model, which consists of the last part of the URI.
hasAlgorithmName	string	Algorithm	The name of the algorithm.
hasAlgorithmDescription	string	Algorithm	Algorithm description.
hasClass	string	Algorithm	The Java class of this algorithm.
hasAppName	string	MiningApp	The name of the mining application.
creator	string	MiningApp	The author of the mining application.
hasName	string	Param	The name of an algorithm parameter.
hasValue	anytype	Param	The value of this algorithm parameter.
hasVarName	string	Feature	The name of the SPARQL variable behind this feature.
hasFeatureType	string	Feature	Defines the content type of this feature.
isPredict	integer	Feature	Indicates whether this feature is to be predicted or not.
linkName	string	Link	Name of a link between two features.

Table 4.4: SPARQL mining ontology datatype properties.

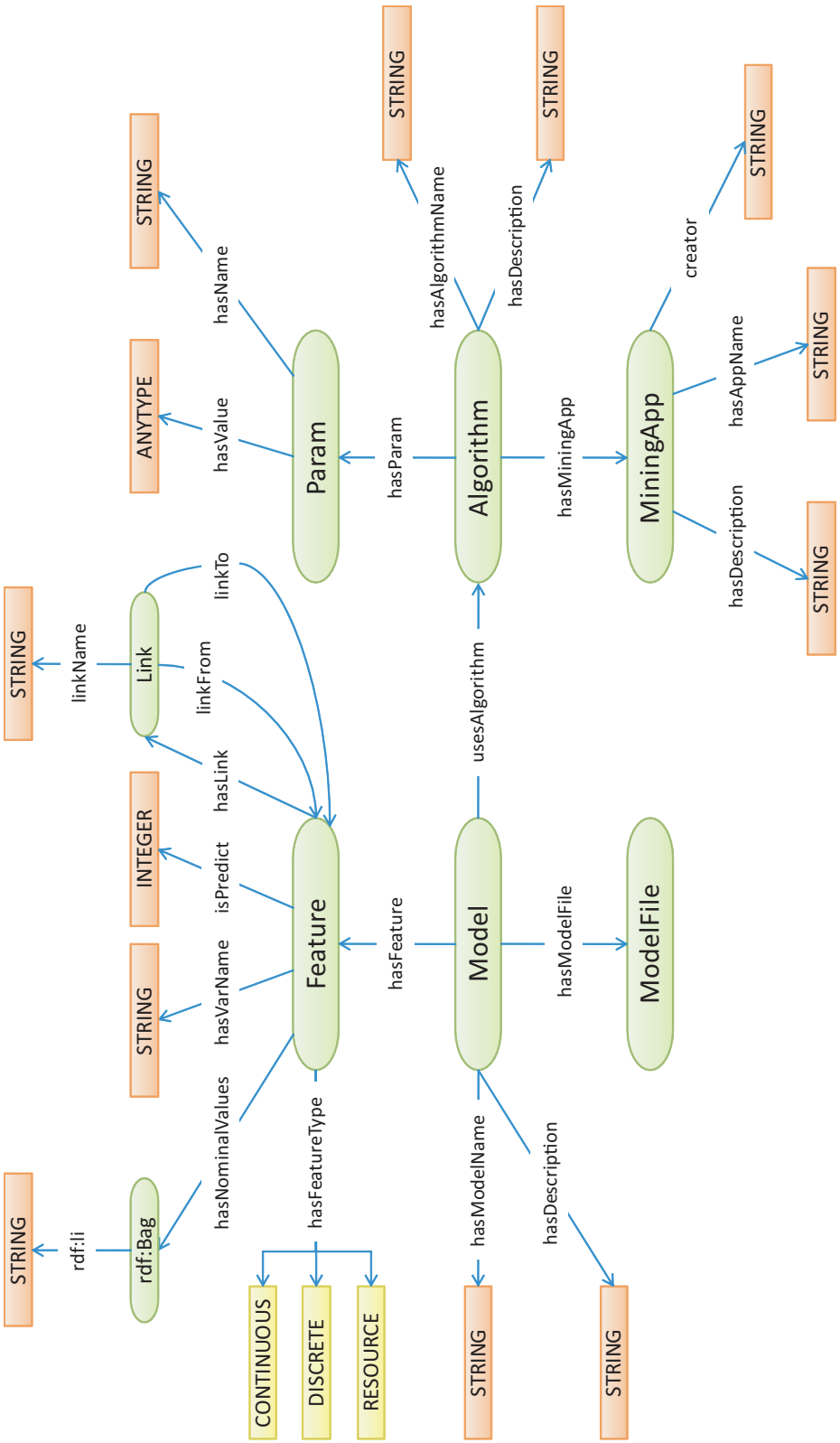


Figure 4.1: The SPARQL mining ontology (SMO) defines the concepts and relations for the metadata of a mining model.

4.3 Making Predictions

After inducing a model with the `CREATE MINING MODEL` statement, SPARQL-ML allows to make predictions with the model via two new property functions (*i.e.*, `sml:mappedPredict` and `sml:predict`).[2] The concept behind this is simple: whenever the predicate of a triple pattern is prefixed with a special name (*i.e.*, `sml`), a call to an external function is made and arguments are passed to the function (by the object of the triple pattern).

The example query in Listing 4.3 explains the usage of `sml:mappedPredict` (line 9). As argument, the function takes the identifier of the previously learned model of the model learning step (Section 4.1) and the instance as specified by the parameters used whilst training the model (in our case specified by the variables $?project$, $?success$, $?member$, and $?class$). In Listing 4.1, we induced a classifier to predict the value for the variable $?success$ on the training data. This classifier is then used on line 9 in Listing 4.3 to predict the value for $?success$ on the test data with a different structure. The result of the prediction, either 'YES' or 'NO', and its probability are finally bound on line 8 to the variables $?prediction$ and $?probability$ respectively. Note that we also defined a shorter version for `mappedPredict` in the case the model is used on a dataset with the same ontology structure (*i.e.*, `predict`; Listing 4.4).

```
1  PREFIX sml: <java:ch.uzh.ifi.sparqlml.mining.pfunction.>
2
3  SELECT DISTINCT ?person ?prediction ?probability
4  WHERE {
5    ?person ex:hasAward ?award .
6    ?person ex:hasFriend ?friend .
7    ?friend rdf:type ?class .
8    ( ?prediction ?probability )
9              sml:mappedPredict ( <http://www.example.org/projectSuccess>
10                                 '?project = ?person'
11                                 '?success = ?award'
12                                 '?member = ?friend'
13                                 '?class = ?class' ) }
```

Listing 4.3: SPARQL-ML `PREDICT` query example 1: apply the model on a dataset with a different ontology structure.

```
1  PREFIX sml: <java:ch.uzh.ifi.sparqlml.mining.pfunction.>
2
3  SELECT DISTINCT ?project ?success ?prediction
4  WHERE {
5    ?project ex:isSuccess ?success .
6    ?project ex:hasTeam ?member .
7    ?member rdf:type ?class .
8    ( ?prediction ?probability )
9              sml:predict ( <http://www.example.org/projectSuccess>
10                           ?project ?success ?member ?class ) }
```

Listing 4.4: SPARQL-ML `PREDICT` query example 2: apply the model on a dataset with the same ontology structure.

[2] `http://jena.sourceforge.net/ARQ/extension.html#propertyFunctions`

By implementing the prediction task as a property function, we are able to use different models in the same SPARQL query by writing multiple predict triples as shown in Listing 4.5. Hence, the comparison of the results predicted by different algorithms in the same query is possible. In our example we apply two models on the same test data, of which the first model was trained with a Relational Probability Tree (line 9) and the second one with a Relational Bayes Classifier (line 12).

Another benefit of this approach is that no further syntax changes have to be introduced into the SPARQL grammar. This approach is called the *virtual triple* approach [Kiefer et al., 2007], as triple patterns including property functions are not matched against the underlying ontology graph, but against the data mining model in this case.

```
1  PREFIX sml: <java:ch.uzh.ifi.sparqlml.mining.pfunction.>
2
3  SELECT DISTINCT ?project ?predictionRPT ?predictionRBC
4  WHERE {
5    ?project ex:isSuccess ?success .
6    ?project ex:hasTeam ?member .
7    ?member rdf:type ?class .
8    ?predictionRPT
9              sml:predict ( <http://www.example.org/projectSuccessRPT>,
10                           ?project ?success ?member ?class ) .
11   ?predictionRBC
12             sml:predict ( <http://www.example.org/projectSuccessRBC>,
13                           ?project ?success ?member ?class ) }
```

Listing 4.5: SPARQL-ML `PREDICT` query example 3: apply different models in the same query.

Semantics of `PREDICT` Queries

The semantics of our `PREDICT` query is basically that of a *prediction join*:[3] (1) `mappedPredict` maps the variables in the Basic Graph Patterns (BGP) to the features in the specified model, which allows us to apply a model on a dataset with a different ontology structure; (2) `mappedPredict` creates instances out of the mappings according to the induced model; (3) the model is used to classify an instance as defined in the `CREATE MINING MODEL` query (Listing 4.1); and (4), the values of the prediction and its probability are bound to variables in the `PREDICT` query (line 8 in Listing 4.3).

[3] `http://msdn2.microsoft.com/en-us/library/ms132031.aspx`

5

Implementation

After having discussed the theoretical concepts of relational data mining on Semantic Web data, we are going to present our implementation of SPARQL-ML to perform knowledge discovery tasks in the Semantic Web. Besides the support of the SPARQL-ML syntax, different propositional and relational learning algorithms should be integrated. In the course of this chapter we explain our approach by first presenting the software we based our implementation on and then describe the design and workflow of our application.

5.1 Software

For the implementation of SPARQL-ML we chose different open-source software tools. The core of our application is built around the query processor *ARQ*[1] for *Jena*.[2] Furthermore, we integrated two mining modules: the open-source machine learning tools *Weka*[3] and *Proximity*[4] that provide a collection of traditional and relational machine learning algorithms.

5.1.1 ARQ/Jena

ARQ is a query engine for Jena that supports the SPARQL RDF Query language described by Prud'hommeaux and Seaborne [Prud'hommeaux and Seaborne, 2007]. Jena is an open-source Java framework for building Semantic Web applications [Carroll et al., 2004]. It contains several tools for RDF, RDFS and OWL, SPARQL, and also provides a rule-based inference engine. More precisely, it allows the use of several inference engines or reasoners, which can be configured to match the users needs. For our experimental evaluation in Chapter 6 we applied the transitive reasoner for OWL that allows inferencing on the transitive and symmetric properties of *rdfs:subPropertyOf* and *rdfs:subClassOf*. Since it only considers these properties and disregards any other definitions, it is orders of magnitude faster than the other OWL reasoners.

[1] `http://jena.sourceforge.net/ARQ/`
[2] `http://jena.sourceforge.net/`
[3] `http://www.cs.waikato.ac.nz/ml/weka/`
[4] `http://kdl.cs.umass.edu/software/`

ARQ makes use of Jena and is easily extendable with custom functions and was therefore our first choice for the implementation of SPARQL-ML. The ARQ query engine provides a full implementation of SPARQL and allows the implementation of new grammar rules.

5.1.2 Weka

The Weka toolkit was developed at the University of Waikato and offers a collection of machine learning algorithms. It focuses on propositional machine learning algorithms and additionally provides tools for data pre-processing and visualization. We chose Weka as a mining module because of its wide spread use in research. The Weka algorithms implemented so far can only handle continuous or nominal values during the model learning process. Hence, the feature input for Weka has to consist of continuous or discrete features with the indication of all possible values. Our framework does not yet support the automatic propositionalization of relational data. Therefore, the algorithms offered by Weka should only be used on data that is already in a simple vector style format.

5.1.3 Proximity

Proximity is an open-source system for relational knowledge discovery designed and implemented by the Knowledge Discovery Laboratory in the Department of Computer Science at the University of Massachusetts Amherst. In order to work with Proximity, the user has to have an instance of MonetDB running before executing any SPARQL-ML queries.[5] With the use of a vertical database like MonetDB, Proximity is orders of magnitude faster than systems hosted on traditional SQL databases for the operations needed by relational data mining.

The data structure used by Proximity to represent the instances of the data mining process is based on labeled graphs in which vertices correspond to objects and edges to links. Because of the given similarities between this data structure and the structure of Semantic Web data introduced in Chapter 2, we decided to integrate Proximity as another mining module.

5.2 Design and Workflow

We chose a modular architecture for the implementation of SPARQL-ML. Our modules represent different data mining tools, which offer numerous learning algorithms for SPARQL-ML. By treating the mining tools as modules, we can easily extend our approach with more data mining tools as they become available. This allows the use of different machine learning techniques with one integrated tool. Our final implementation of SPARQL-ML is an extension of ARQ, which can be used with an existing ARQ installation.

Figure 5.1 presents the workflow for data mining on Semantic Web data with our SPARQL-ML implementation. In order to explain the elements of our extension, we will walk through the steps in Figure 5.1. (1) A SPARQL-ML query is sent to the *Query Engine* of ARQ, where a *QueryML* object is created that stores the specified information of the query. (2) The query engine performs the matching of the basic graph patterns to the underlying data and creates

[5] `http://monetdb.cwi.nl/`

the mappings for the variables. (3) The *ResultSet* is then passed on to the *Mining Module* that implements the requested algorithm. (4) The *Mining Module* first transforms the *ResultSet* into our internal subgraph structure (Section 5.3). When learning a new model, the *Mining Module* (5) serializes the *QueryML* object to RDF after the definitions of our *SPARQL Mining Ontology*. When performing a prediction task, the *Mining Module* (5) accesses the metadata of the specified mining model. Finally, the *Mining Module* (6) returns either the metadata of the mining model when learning or the predicted values when predicting. Appendix C shows a Java code example on how to access the described functions of SPARQL-ML.

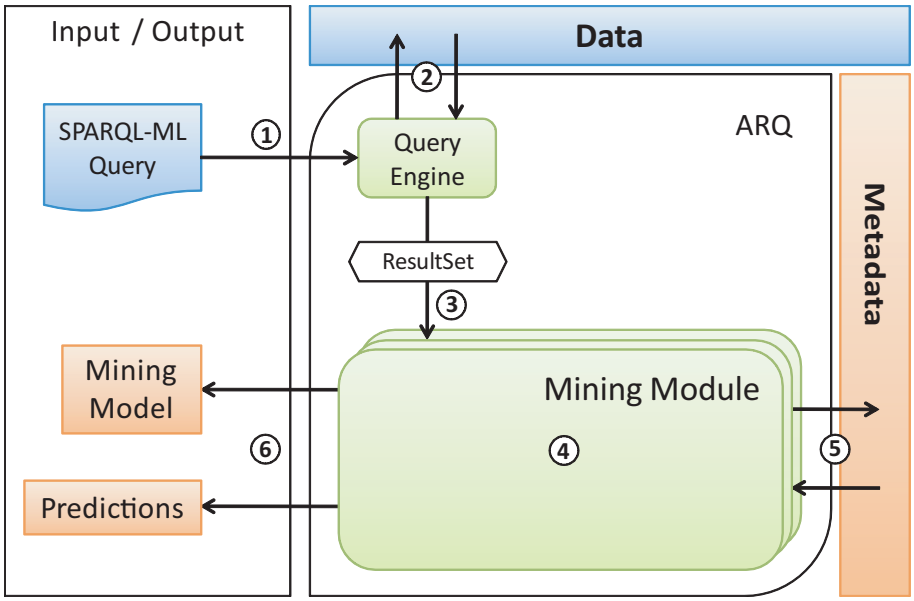


Figure 5.1: Workflow of our SPARQL-ML implementation.

5.3 Subgraph Data Structure

In order to identify the single instances in a dataset, our implementation transforms the ARQ *ResultSet* into a generic subgraph data structure for further processing. It follows a simple subgraph pattern: each subgraph consists of one or more objects (*resources*), links (*properties*) and attributes (*literals*). Each subgraph contains exactly one target object for which we want to make a prediction. The object is specified with the `TARGET` keyword in the `CREATE MINING MODEL` query (see Section 4.1).

Listing 5.1 shows the pseudo algorithm for the transformation of the *ResultSet* into our subgraph structure. Its output is a collection of subgraphs for the use with the different algorithms available in our implementation. As input, it requires the *ResultSet* of the data and the *FeatureSet* of the model. The *FeatureSet* defines the structure of the subgraph and consists of the specified variables in the `CREATE MINING MODEL` query (see Chapter 4). It is, therefore, stored in the metadata. For the detection of the single subgraphs, we iterate through the results and treat the

target object as the root node of the subgraph. The remaining features of the *ResultSet* with a direct or indirect link to the determined root node belong to the same subgraph.

Once the subgraphs are stored in the generic data structure, we can easily export the data into any given format for the use with the implemented data mining modules. Our SPARQL-ML implementation performs an export to either the *Proximity XML*[6] format or the *Weka ARFF*[7] format.

```
1 Outputs: subgraphs, a collection of subgraphs
2
3 Inputs:
4 resultSet ←the set of all variable mappings;
5 featureSet ←the set of features chosen in the create mining model query;
6
7 s ← ∅;
8 subgraphs ←{s};
9
10 for each result res in resultSet do
11  if subgraphs not contains res.Root then
12   s.addObject(res.Root);
13   subgraphs.put(s);
14  endif
15  for each feature currFeat in featureSet do
16   if res.currFeat.isURI() then
17    s.addObject(res.currFeat);
18   else if res.currFeat.isBlank() then
19    s.addObject(res.currFeat);
20   else if res.currFeat.isLiteral() then
21    s.addAttribute(res.currFeat);
22   endif
23  endfor
24 endfor
```

Listing 5.1: Pseudo algorithm for the transformation of a SPARQL resultset into a generic subgraph structure.

5.4 Package Structure

Figure 5.2 illustrates the main Java packages of our implementation, which is divided into an *arq* and a *mining* package. The first one extends the query engine of ARQ, while the second one provides interfaces for the interaction with different mining modules. The persistent storage of the data mining models and accompanying metadata is organized in a *mining* folder inside the ARQ root directory.

[6] http://kdl.cs.umass.edu/proximity/documentation/tutorial/apc.html
[7] http://weka.sourceforge.net/wekadoc/index.php

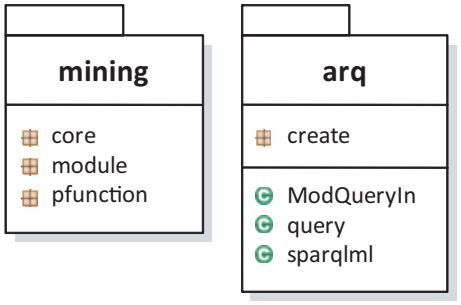


Figure 5.2: UML diagram of the main packages of the SPARQL-ML extension for ARQ.

5.4.1 The ARQ Package

The *arq* package implements the syntax extensions of SPARQL-ML. For that purpose, the parser had to be adapted with the new syntax introduced in Chapter 4. The ARQ parser was built with the help of the Java Compiler Compiler (JavaCC),[8] a parser generator for the translation of grammar files into Java code that recognizes matches to a given grammar. We present our full grammar extensions to the ARQ parser in Appendix A.

5.4.2 The Mining Package

The *mining* extension in Figure 5.3 provides the methods and tools to enable data mining for knowledge discovery in Semantic Web data. The *core* package contains the functionality to transform the data returned by the query process into our internal data structure as described in Section 5.3. The *pfunction* package contains the property functions of the `PREDICT` query introduced in Section 4.3. Since a property function represents a traditional way of extending ARQ, we did not place the package into the *arq* package. Finally, the *module* package contains the available mining modules. Each mining module must have a class that implements the interface *MiningApp*.

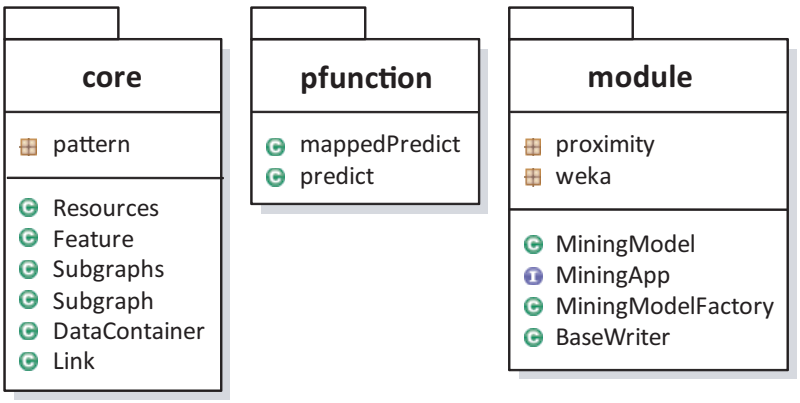


Figure 5.3: UML diagram of the packages inside the mining extension.

[8] `https://javacc.dev.java.net/`

The interface defines the following methods:

- `init(MiningModel m)`: initializes the mining module with a given mining model.
- `prepareModelData(ResultSet res, HashMap<Var, Var> mappings)`: provides the necessary data preparation functions for the mining module.
- `learnModel()`: learns a new model with a given learning algorithm.
- `applyModel()`: applies an existing model on new data to get a prediction.
- `getClassPrediction(int subgID)`: return the predicted value of a given subgraph.
- `getClassProbability(int subgID)`: return the probability of the prediction given by *getClassPrediction*.

Our extension can communicate with every data mining module that implements the given interface. Each mining module also contains a class that extends the class *BaseWriter*, which provides the functionality to export our internal subgraph data structure into the appropriate format for the mining tool. In addition, the mining package implements the interfaces to the learning algorithms and the helper classes for the evaluation of induced models.

Algorithms

Our implementation supports several different relational and propositional learning algorithms provided by *Proximity* and *Weka*. In order to use an algorithm, it has to be described with the necessary metadata following our *SPARQL Mining Ontology* (Section 4.2). We implemented the following algorithms so far:

- **Weka**

  J48: a tree learner based on the C4.5 decision tree.

  M5P: a tree learner for continuous classes.

  Linear Regression: a linear regression for prediction.

- **Proximity**

  Relational Bayesian Classifier: a simple bayesian classifier for relational data.

  Relational Probability Tree: a tree learner for relational data.

The full RDF metadata of the implemented algorithms and the associated mining modules is described in Appendix B.

Evaluation Features

In order to review and judge predictions made by SPARQL-ML, we implemented an evaluation feature to compare the performance of learned models with several standard data mining evaluation methods. Every prediction automatically saves its evaluation results as a text file in the */mining/results* subfolder of ARQ. Besides the accuracy and the confusion matrix, the results also contain the points of the ROC-curves and the area under the ROC-curve. We used these measures to compare the results of our experiments in Chapter 6.

6

Evaluation

The goal of our evaluation was to show the usefulness and the simplicity of the integration of machine learning methods with the existing Semantic Web infrastructure. Furthermore, we wanted to show that the combination of logic deduction and statistical induction holds over induction only. To that end, we conducted four experiments:

1. **Relational prediction experiment:** test of our implementation on a simple relational dataset;

2. **Business project success prediction experiment:** evaluation of the prediction quality in a synthetic dataset;

3. **Semantic Web service domain prediction experiment:** estimation of the classification performance using a Semantic Web service dataset;

4. **SVM-Benchmarking experiment:** comparison of our approach with an approach based on kernel methods used in Support Vector Machines.

In the following, we describe each of the experiments in detail giving insights into our experimental setup as well as a discussion of the empirical results. We conducted all our experiments on a two processor dual core 2.0GHz machine with 16GB RAM, 7200rpm disks, using a 64bit operating system. Due to the advantages of relational learning methods for Semantic Web data, which we described in detail in Chapter 3, we primarily used the *Proximity* mining module for our evaluation.

6.1 Relational Prediction Experiment

We first conducted a simple prediction experiment to evaluate the applicability of relational learning methods for Semantic Web data. It is not focused on achieving superior results, but rather serves as a proof of concept and illustrates the capabilities of SPARQL-ML. Because of the limited possibilities of our dataset, we did not use an ontology for inferencing.

6.1.1 Evaluation Procedure and Dataset

Our dataset contains descriptions about movies from the Internet Movie Database (IMDB).[1] The IMDB provides its data as raw text files that can be imported into various relational databases. We converted this data to RDF and created a dataset that contains information about movies, actors, directors, producers, studios, and their relations. We concentrated on movies released in the United States from 1996 to 2001. We then split the data into a training and a test set according to the following characteristics:

- Training Set: 1220 movies released in the United States from 1996 to 1999.
- Test Set: 736 movies released in the United States from 2000 to 2001.

Each movie contains an attribute indicating the opening-weekend receipt information. We discretized this attribute, creating a positive class label for movies with a revenue of more than $2 million during the opening-weekend. The movies with less than $2 million during the opening-weekend received a negative class label. The prior probability of a movie having a positive class is 36%.

6.1.2 Step 1: Learning

We trained two classifiers on our test set, namely a Relational Bayes Classifier (RBC) and a Relational Probability Tree (RPT). Both classifiers are able to predict the discretized class label $?class$. The SPARQL-ML query for the model learning process is shown in Listing 6.1. To ensure that every subgraph has enough structural information, we only used movies that have at least one associated actor, director, producer, and studio.

```
1  PREFIX imdb:    <http://www.imdb.com/>
2
3  CREATE MINING MODEL <http://www.example.org/models/imdb> {
4   ?movie RESOURCE TARGET
5   ?class DISCRETE PREDICT {'YES','NO'}
6   ?genre DISCRETE
7   ?actor RESOURCE
8   ?director RESOURCE
9   ?producer RESOURCE
10  ?studio RESOURCE
11 }
12 WHERE {
13  ?movie imdb:hasClass ?class .
14  ?movie imdb:hasGenre ?genre .
15  ?movie imdb:hasActor ?actor .
16  ?movie imdb:hasDirector ?director .
17  ?movie imdb:hasProducer ?producer .
18  ?movie imdb:hasStudio ?studio .
19 } USING <http://kdl.cs.umass.edu/proximity/rbc>
```

Listing 6.1: SPARQL-ML create query for IMDB box office revenue prediction.

[1] http://www.imdb.com

In order to evaluate the correct representation of our resulting instances for the use in the model learning process, we visualized a few example subgraphs created by the SPARQL-ML query. The example subgraphs in Figure 6.1 clearly show the heterogeneous structure of our data: a movie must have at least one association with an actor, director, producer, and studio, but is allowed to have more than one. A propositional learning algorithm must have a fix amount of attributes to train a model on such data, whereas a relational learning algorithm can deal with an arbitrary amount of attributes and objects for each subgraph. This is possible due to the automatic flattening of the data as introduced in Section 3.3.1.

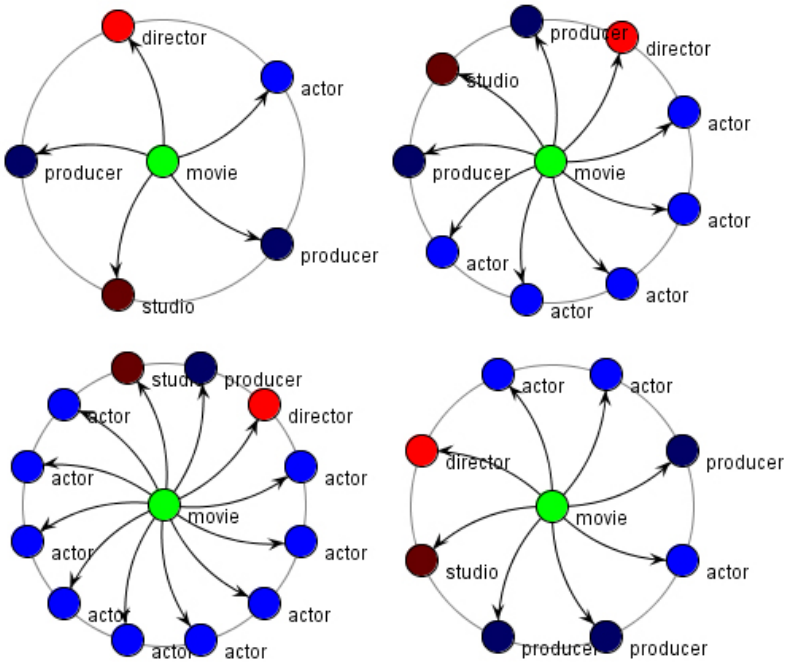


Figure 6.1: IMDB instances represented as subgraphs.

6.1.3 Step 2: Prediction

After having successfully learned our models, we evaluated our classifiers on our test set with the prediction query shown in Listing 6.2.

6.1.4 Results and Discussion

The Figure 6.2 shows the results of our experiment in terms of prediction accuracy (ACC; in legend), Receiver Operating Characteristics (ROC; graphed), and the area under the ROC-curve (AUC; in legend). The Receiver Operating Characteristics (ROC-curve) graphs the true positive rate (y-axis) against the false positive rate (x-axis), where an ideal curve would go from the origin to the top left corner (0,1), before proceeding to the top right one (1,1) [Provost and Fawcett, 2001].

```
1  PREFIX imdb:     <http://www.imdb.com/>
2  PREFIX sml:      <java:ch.uzh.ifi.sparqlml.mining.pfunction.>
3
4  SELECT DISTINCT ?movie ?class ?prediction
5  WHERE {
6   ?movie imdb:hasClass ?class .
7   ?movie imdb:hasGenre ?genre .
8   ?movie imdb:hasActor ?actor .
9   ?movie imdb:hasDirector ?director .
10  ?movie imdb:hasProducer ?producer .
11  ?movie imdb:hasStudio ?studio .
12  ?prediction
13          sml:predict ( <http://www.example.org/models/imdb>
14                        ?movie ?class ?genre ?actor ?director
15                        ?producer ?studio )
16 }
```

Listing 6.2: SPARQL-ML predict query for the IMDB box office revenue prediction.

It serves as a prior-independent approach for comparing the quality of a predictor. The area under the ROC-curve is, typically, used as a summary number for the curve.

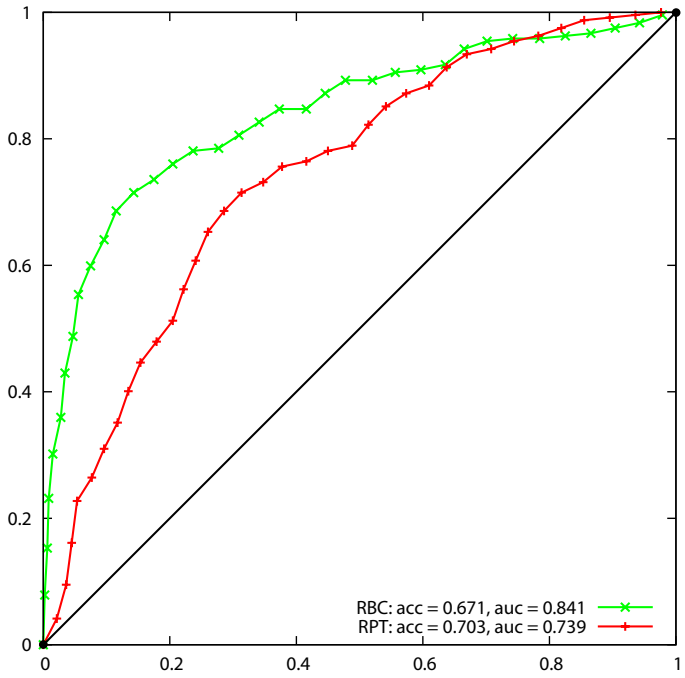


Figure 6.2: ROC-curves of the IMDB box office revenue prediction.

The learning algorithms shown are a Relational Probability Tree (RPT) and a Relational Bayesian Classifier (RBC). The RPT has a slightly better accuracy than the RBC, but has a lower score for

the area under the ROC-curve. Hence, we conclude that the RBC is a better choice for our movie box-office prediction task. All in all, the results are not that impressive, which is why a future experiment would have to include additional attributes in the model learning task, such as the age of an actor, to achieve a better prediction quality.

6.2 Project Success Prediction Experiment

In order to demonstrate the enhancement of the data mining task with inferred data, we conducted an experiment with a synthetic dataset. Due to the synthetic nature, we hope to better understand the results and reduce any experimental noise, such as different correlations between attributes and the class label. The goal of this experiment is to predict, whether a business project will be successful or not.

6.2.1 Evaluation Procedure and Dataset

The constructed *Business Project dataset* consists of different business projects and the employees of an imaginary company. The company occupies 40 employees, of which everyone has exactly one out of 8 different occupations. Figure 6.3 shows part of the created business ontology in more detail. 13 employees belong to the superclass `Manager`, whereas 27 employees belong to the superclass `Non-Manager`.

We then created business projects and randomly assigned up to 6 employees to each project. The resulting teams consist of 4 to 6 members. As a final step we randomly defined each project to be successful or not, with a bias for projects being more successful, if more than three team members are of the type `Manager`. We did not add any attributes to the employees to avoid a possible correlation between the project outcome and the attributes. The resulting dataset contains 400 projects with different teams. The prior probability of a project being successful is 35%. We did a 50:50 split of the data and followed a single holdout procedure, swapping the roles of the testing and training set and averaged the results. We applied a stratification technique to preserve the original distribution of class labels in both sets.

6.2.2 Step 1: Learning

The only available feature for the learning process is the information given through the employees assigned to the projects. Our target attribute for the prediction is the variable $?success$. Since the target attribute can either have the value 'YES' or 'NO', we learned a binary classifier. Listing 6.3 shows the whole SPARQL-ML create query for the model learning process.

We learned different classifiers to compare the performance of different learning algorithms on our dataset. We compare the results of an RPT, an RBC and a k-Nearest-Neighbor algorithm. As another varying parameter, we once learned a classifier without inferred triples for every algorithm and once with additional inferred triples. We use the ontology illustrated in Figure 6.3, which we presented in Chapter 3.

With the Jena reasoner disabled, the last triple (line 10) of the basic graph pattern in Listing 6.3 matches only the directly instantiated class of an employee. Therefore, the learning algorithm does not receive any inferred triples as additional features. With the reasoner enabled, the same

```
CREATE MINING MODEL <http://www.example.org/models/projectSuccess> {
 ?project RESOURCE TARGET
 ?success DISCRETE PREDICT {'YES','NO'}
 ?member RESOURCE
 ?class RESOURCE
}
WHERE {
 ?project ex:isSuccess ?success .
 ?project ex:hasTeam ?member .
 ?member rdf:type ?class .
}
USING <http://kdl.cs.umass.edu/proximity/rpt>
```

Listing 6.3: SPARQL-ML create query for the business project success prediction.

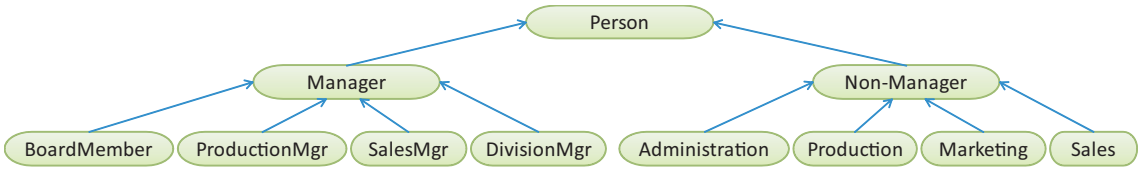


Figure 6.3: Example business ontology.

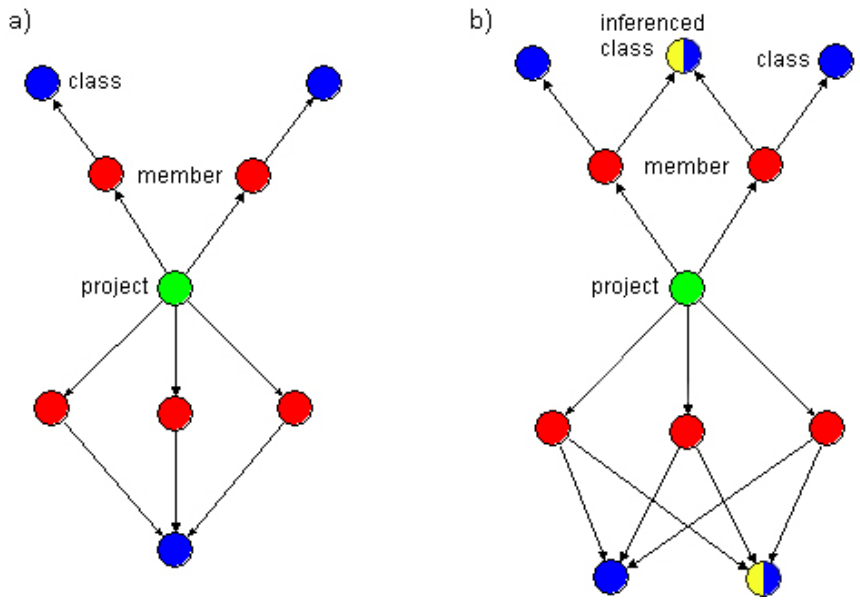


Figure 6.4: Example subgraphs built from data a) without inferencing and b) with inferencing.

triple matches also the newly inferred triples, indicating whether an employee belongs to the `Manager` or to the `Non-Manager` class. Figure 6.4 shows two abstract subgraphs resulting from our SPARQL-ML query. The subgraph in Figure 6.4.a) contains only the explicit information

given in our dataset, which consists of the direct types of an employee. The subgraph in Figure 6.4.b) is extended with additional inferred triples, which have been made explicit. As a result, the subgraph contains more information although we used exactly the same dataset.

6.2.3 Step 2: Prediction

The prediction query shown in Listing 6.4 outputs the distinct projects, their actual value for the property $isSuccess$, and the prediction given by our classifier. We again enabled the reasoner for the use with our mining model that was learned with the additional inferred data.

```
1 PREFIX sml: <java:ch.uzh.ifi.sparqlml.mining.pfunction.>
2
3 SELECT DISTINCT ?project ?success ?prediction
4 WHERE {
5  ?project ex:isSuccess ?success .
6  ?project ex:hasTeam ?member .
7  ?member rdf:type ?class .
8  ?prediction
9          sml:predict ( <http://www.example.org/models/projectSuccess>
10                       ?project ?success ?member ?class )
11 }
```

Listing 6.4: SPARQL-ML predict query for the business project success prediction.

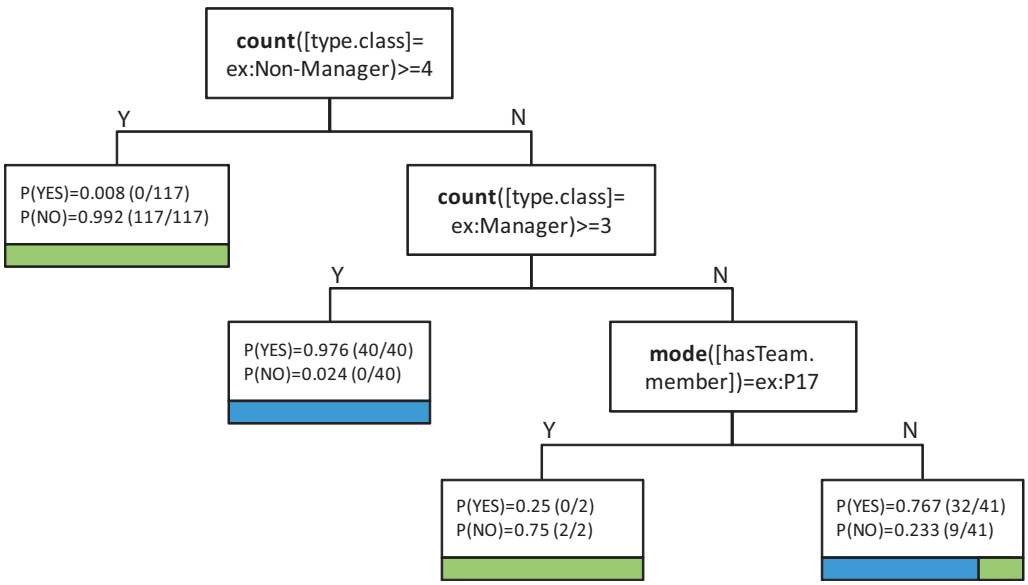


Figure 6.5: RPT of the business project success prediction experiment (with inferencing support).

6.2.4 Results and Discussion

Due to the synthetic data it is quite obvious that without inferencing the model learning algorithm will not have access to the ontological information, whether an employee is a manager or not, and hence, should score worse. The results shown in Figure 6.6 confirm our assumptions. We present the results of a Relational Probability Tree (RPT), a Relational Bayes Classifier (RBC), both with and without inferencing, and as a baseline, we applied a k-Nearest Neighbor (k-NN) learning algorithm with inferencing and k=9, using a maximum common subgraph isomorphism metric [Valiente, 2002]. The k-NN belongs to the supervised learning algorithms and basically compares every subgraph in the training set with every subgraph in the test set. For every instance in the test set, we then only consider the top 9 subgraphs from the training set with the highest similarity. We then used the most common class amongst the nearest neighbors as our final prediction. In addition to the accuracy as a measure for the quality of the prediction, we also calculated the ROC-curves shown in Figure 6.6, which show the sensitivity and specificity of the binary classifier. We present the area under the ROC-curve as an additional measure for comparing the ROC-curves.

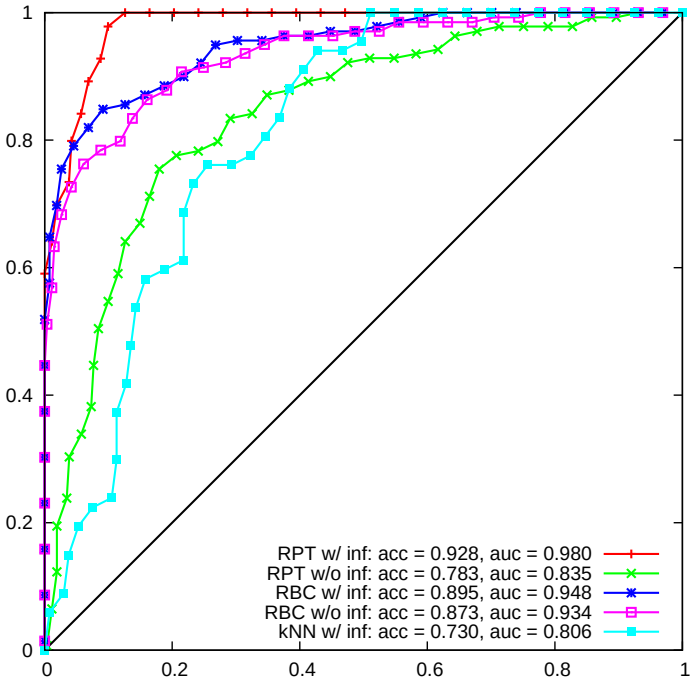


Figure 6.6: ROC-Curves of business project success prediction.

Figure 6.6 shows that the RPT with inferencing was able to outperform all other learning algorithms concerning accuracy and AUC. The RPT has also the biggest differences in accuracy and AUC between the results with and without inferencing. Therefore, we investigate the structure and features of both RPTs to find the differences.

Figure 6.5 shows the RPT learned with inferencing support. The decision tree is able to detect the best predictor provided by the inferred triples and performs the first and second binary split on the information on how many employees are managers or non-managers.

The first feature splits the 200 subgraphs in our training set into 117 subgraphs with the class

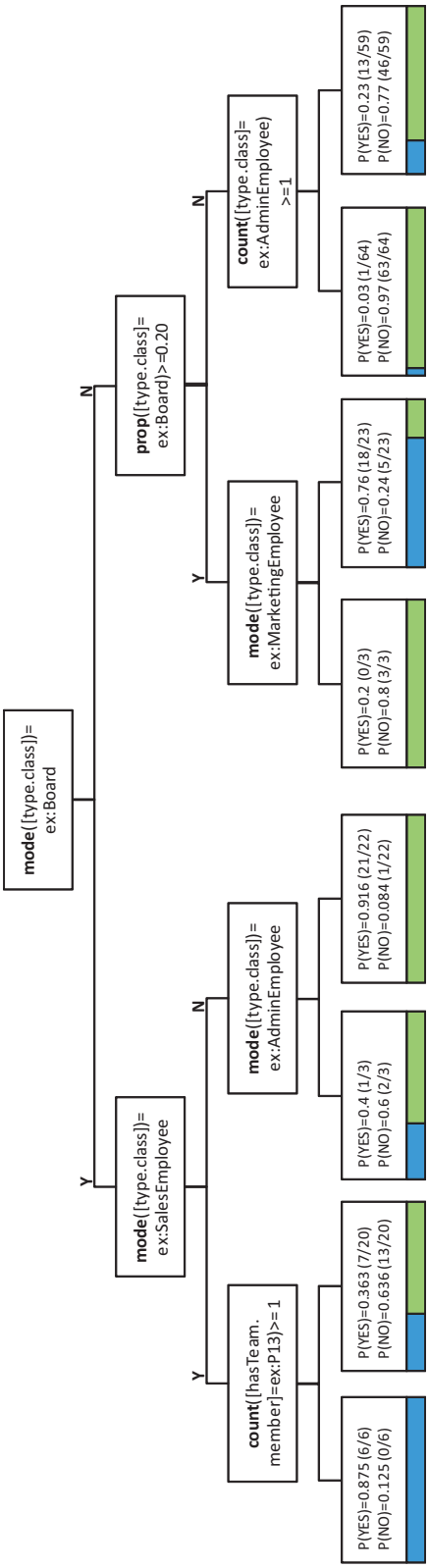


Figure 6.7: RPT of the business project success prediction experiment (without inferencing support).

'NO' and 83 subgraphs with the classes 'YES' and 'NO'. The left branch leads to a leaf since no further split is necessary. According to the computation in Section 3.3.1, the probability of a project being successful at that specific leaf is $\frac{117+1}{117+2} = 0.992$. The remaining 83 subgraphs move down the right branch where they get split again, leading to another leaf with 40 subgraphs with the class 'YES'. The remaining 43 subgraphs are split a last time leading to the two final leafs of the tree. The leafs of the RPT indicate the probability that a subgraph will be of a given class at that specific leaf. In contrast to the expressiveness of the RPT with inferencing, the RPT built without inferencing support in Figure 6.7 is bigger and less selective. The probabilities indicated at the leafs are lower than the probabilities in Figure 6.5 and therefore less reliable. Hence, the structure of the RPTs confirms that the enhancement of the data mining process with inferred triples is able to outperform the same task without inferencing.

6.3 Semantic Web Service Domain Prediction Experiment

With our first set of experiments, we showed how our approach can be used for instance-based prediction on a synthetic dataset. We now apply the same techniques to assess the ability of our approach to correctly classify Semantic Web services in a real-world dataset.

Our dataset consists of Web services that use the OWL-S ontology to describe the concepts and their relations to each other.[2] OWL-S serves as a semantic mark up language for Semantic Web services. According to the creators, the main motivation for the creation of this new ontology were the following:

- Automatic Web service discovery.
- Automatic Web service invocation.
- Automatic Web service composition and interoperation.

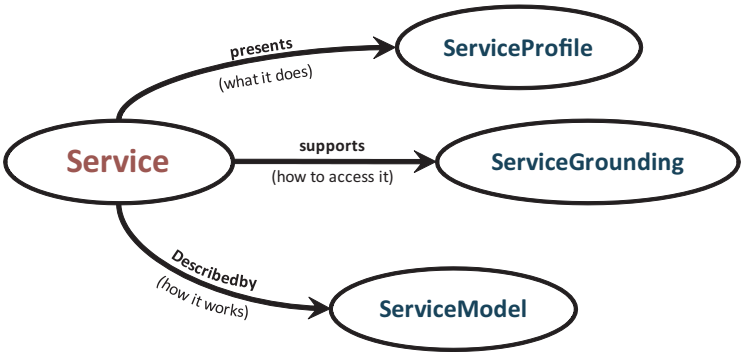


Figure 6.8: Top level of the service ontology after Martin *et al.* [Martin et al., 2004].

We expand the use of the dataset with a scenario that automatically classifies Web services. In order to understand the feature selection for the learning process, it is necessary to introduce

[2] http://www.daml.org/services/owl-s/1.1/

the basic structure of OWL-S in Figure 6.8. A service has a name and a text-description. The service profile helps to identify what the service does, while the service grounding gives information about how an agent can access the service, and the service model tells the user how the service works. For our experiment we concentrate only on the service profile, which provides information about the Input/Output-parameters of the service as we believe that they are most informative for this task.

6.3.1 Evaluation Procedure and Dataset

We used the OWLS-TC v2.1 service retrieval test collection.[3] We did not perform Semantic Web service matchmaking, which is the main goal of the dataset, but used it to perform classification of the given Semantic Web services into 7 different domains. Table 6.1 lists the domains and their prior distribution in the dataset. The dataset contains a total of 578 Web service descriptions in OWLS-S.

We performed a 10-fold cross validation. 90% of the data was used to learn a classification model, and the remaining 10% to test the effectiveness of the learned model. This approach is standard practice in Machine Learning. We again applied a stratification technique to preserve the original distribution of class labels in all 10 sets.

Service domain	Distribution (%)
weapon	4.33
food	4.33
communication	5.02
medical	8.99
travel	18.34
education	23.36
economy	35.63

Table 6.1: OWLS-TC service domain distribution.

6.3.2 Step 1: Learning

We first learned a Relational Probability Tree without inferencing and then enabled inferencing for a second run. We defined the variable $?domain$ as our target attribute to be predicted. The service profile with its input and output parameters serves as the available information for the model learning process. Listing 6.5 shows the complete SPARQL-ML query used to learn the classifier. By using `OPTIONAL` graph patterns it is possible to include services with no outputs or inputs in the learning process. The additional `OPTIONAL` graph pattern for the `rdfs:subClassOf` triples (lines 25, 30) enables us to run the same query with and without the support of an ontology. With inferencing enabled, the SPARQL query finds additional matches for the OPTIONAL triple patterns with the property `rdfs:subClassOf` by inferring supplementary types of service in- and outputs.

[3] `http://projects.semwebcentral.org/projects/owls-tc/`

Figure 6.9 visualizes the abstract subgraphs created by our `CREATE MINING MODEL` query. Figure 6.9.a) represents the subgraph without inferred triples and 6.9.b) shows the additional inferred triples $outputSuper$ and $inputSuper$ for the subgraph.

```
1  PREFIX rdfs:    <http://www.w3.org/2000/01/rdf-schema#>
2  PREFIX service: <http://www.daml.org/services/owl-s/1.1/Service.owl#>
3  PREFIX profile: <http://www.daml.org/services/owl-s/1.1/Profile.owl#>
4  PREFIX process: <http://www.daml.org/services/owl-s/1.1/Process.owl#>
5
6  CREATE MINING MODEL <http://www.ifi.uzh.ch/ddis/services> {
7   ?service RESOURCE TARGET
8   ?domain DISCRETE PREDICT
9    {'communication','economy','education',
10    'food','medical','travel','weapon'}
11  ?profile RESOURCE
12  ?output RESOURCE
13  ?outputType RESOURCE
14  ?outputSuper RESOURCE
15  ?input RESOURCE
16  ?inputType RESOURCE
17  ?inputSuper RESOURCE
18 }
19 WHERE {
20  ?service service:presents ?profile .
21  ?service service:hasDomain ?domain .
22  OPTIONAL {
23   ?profile profile:hasOutput ?output .
24   ?output process:parameterType ?outputType .
25   OPTIONAL { ?outputType rdfs:subClassOf ?outputSuper . }
26  }
27  OPTIONAL {
28   ?profile profile:hasInput ?input .
29   ?input process:parameterType ?inputType .
30   OPTIONAL { ?inputType rdfs:subClassOf ?inputSuper . }
31  }
32 }
33 USING <http://kdl.cs.umass.edu/proximity/rpt> ('maxDepth'=6)
```

Listing 6.5: SPARQL-ML create query of the Semantic Web service classification.

6.3.3 Step 2: Prediction

The prediction query shown in Listing 6.6 outputs the distinct services, their correct domain, and the prediction given by our classifier. We again enabled the reasoner for the use with our mining model that was learned with the additional inferred data.

```
PREFIX rdfs:    <http://www.w3.org/2000/01/rdf-schema#>
PREFIX service: <http://www.daml.org/services/owl-s/1.1/Service.owl#>
PREFIX profile: <http://www.daml.org/services/owl-s/1.1/Profile.owl#>
PREFIX process: <http://www.daml.org/services/owl-s/1.1/Process.owl#>
PREFIX sml:     <java:ch.uzh.ifi.sparqlml.mining.pfunction.>

SELECT DISTINCT ?service ?domain ?prediction
WHERE {
 ?service service:presents ?profile .
 ?service service:hasDomain ?domain .
 OPTIONAL {
  ?profile profile:hasOutput ?output .
  ?output process:parameterType ?outputType .
  OPTIONAL { ?outputType rdfs:subClassOf ?outputSuper . }
 }
 OPTIONAL {
  ?profile profile:hasInput ?input .
  ?input process:parameterType ?inputType .
  OPTIONAL { ?inputType rdfs:subClassOf ?inputSuper . }
 }
 ?prediction
         sml:predict ( <http://www.ifi.uzh.ch/ddis/services>
                       ?service ?profile ?domain ?output ?outputType
                       ?outputSuper ?input ?inputType ?inputSuper )
}
```

Listing 6.6: SPARQL-ML predict query of the Semantic Web service classification.

6.3.4 Results and Discussion

The averaged classification accuracy of the results of the 10 runs are given in Table 6.2. Through inferencing we were able to improve the accuracy by 0.319. The more detailed results in Table 6.3 show even better that the model learning algorithm is able to improve the results for all 7 domains present in the dataset. We used the well-known data mining measures False Positives Rate (FP Rate), Precision, Recall and F-measure that is the weighted harmonic mean of precision and recall, to present the class-specific improvements. As the results of a paired one-tailed t-test show, the differences for the Recall and F-measure are highly significant. The results for Precision just barely misses significance at the 95% level.

	Accuracy
w/o Inferencing	0.5102
w/ Inferencing	0.8288

Table 6.2: Accuracy for the Semantic Web service classification.

A closer look at the structure and binary features used in the RPT with inferencing, reveals that the features are always computed with the help of the `rdfs:subClassOf` predicate, which

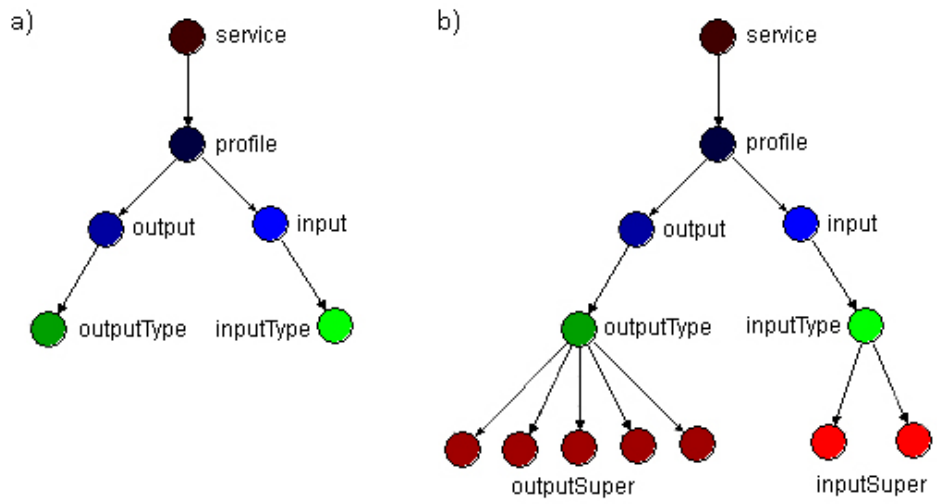


Figure 6.9: Subgraphs of the Semantic Web service classification task, a) without inferencing and b) with inferencing.

speaks for the good predictive quality of the underlying concept hierarchy. An example RPT is shown in Appendix D. Not less than 6 features are found inside the Suggested Upper Merged Ontology (SUMO) [Pease et al., 2002] that is extremely broad in scope and serves as a generalized upper ontology. Through the access to these newly inferred triples, the learning algorithm is able to find corresponding super-classes for several domains. Table 6.4 lists some of the inferred types that serve as binary features in the decision tree shown in Appendix D. New features with better support for the domains *food*, *travel* and *medical* where generated. All the Semantic Web services inside the weapon domain deal with the funding, giving or financing of weapons, which is why the super class `ChangeOfPossession` is the best match to accurately identify these services. The SUMO ontology did not provide any general predictors for the classification of the remaining domains.

Domain	**FP Rate**		**Precision**		**Recall**		**F-measure**	
	w/o inf	*w/ inf*	*w/o inf*	*w/ inf*	*w/o inf*	*w/ inf*	*w/o inf*	*w/ inf*
weapon	0.002	0.002	0.917	0.964	0.367	0.900	0.524	0.931
food	0	0.002	0	0.960	0	0.800	0	0.873
communication	0.007	0.004	0.819	0.900	0.600	0.600	0.693	0.720
medical	0.006	0.030	0	0.688	0	0.550	0	0.611
travel	0	0.069	1	0.744	0.245	0.873	0.394	0.803
education	0.538	0.090	0.311	0.716	0.904	0.869	0.463	0.786
economy	0.081	0.018	0.810	0.964	0.644	0.889	0.718	0.925
Average	**0.091**	**0.031**	**0.551**	**0.848**	**0.394**	**0.783**	**0.399**	**0.807**
t-test (paired, one-tailed)		**p=0.201**		**p=0.053**		**p=0.009**		**p=0.004**

Table 6.3: Detailed, class-specific results for the Semantic Web service domain prediction.

Service domain	Super class of input/output parameter
food	SUMO.owl#Food
medical	SUMO.owl#BiologicalProcess
weapon	SUMO.owl#ChangeOfPossession
travel	travel.owl#Destination

Table 6.4: Detected super classes for the classification of service domains

6.4 SVM-Benchmark Experiment

The final experiment also deals with a real-world Semantic Web dataset containing information about research communities and relations among them. The goal was to replicate the experiments described by Bloehdorn and Sure [Bloehdorn and Sure, 2007] and to compare the results. In their work, they introduce a framework for the design and evaluation of kernel methods that are used in Support Vector Machines, such as SVM^{light}.[4] The framework provides various kernels for the comparison of classes as well as datatype and object properties of instances. Moreover, it is possible to build customized, weighted combinations of such kernels.

Our experiments, therefore, include two tasks: (1) prediction of the affiliation a person belongs to (*person2affiliation*), and (2) prediction of the affiliation a publication is related to (*publication2affiliation*).

6.4.1 Evaluation Procedure and Dataset

The dataset we used is based on the SWRC ontology,[5] which is a collection of OWL annotations for persons, publications, and projects, and their relations from the University of Karlsruhe.[6] The data consists of instances of the type Person, Publication, Topic, Project, and ResearchGroup. 177 instances have type Person with an affiliation to one of the four research groups, whereas 1155 instances have type Publication. Table 6.5 shows the prior distribution of the research group affiliations of persons and publications. For the *person2affiliation* task, the nodes and relations we used for the remaining subgraph contain the `worksAtProject` and `worksAtTopic` object properties. We also include the publication object property pointing to the publications of a given person. We added the `worksAtTopic` object property of the Person class and used it instead of the `workedOnBy` object property of the Topic class to simplify the relations of the resulting subgraphs. For the *publication2affiliation* task, we used the `isAbout` and `author` object properties pointing to associated topics and authors respectively. In order to predict the affiliation a publication is related to, we defined the affiliation to be the research group, where the major part of the authors of this publication belong to. Bloehdorn and Sure defined the affiliation to be the research group where one of the authors is affiliated with.

[4] http://svmlight.joachims.org/
[5] Semantic Web for Research Communities available at http://ontoware.org/projects/swrc/
[6] http://www.aifb.uni-karlsruhe.de/about.html

Research Group	Persons	Publications
Business Information and Comm. Systems (id1)	73	152
Efficient Algorithms (id2)	28	121
Knowledge Management (id3)	60	839
Complexity Management (id4)	16	43
Total	**177**	**1155**

Table 6.5: Distribution of research group affiliations in the SWRC metadata.

6.4.2 Step 1: Learning

We applied an RBC and an RPT learning algorithm to both tasks. The full SPARQL-ML query used for the *person2affliation* prediction is shown in Listing 6.7. We again used the `OPTIONAL` clause for the object properties to include persons without any relations to projects, topics, or publications.

```
1 PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2 PREFIX swrc: <http://swrc.ontoware.org/ontology#>
3
4 CREATE MINING MODEL <http://www.ifi.uzh.ch/ddis/affiliations> {
5  ?person RESOURCE TARGET
6  ?group RESOURCE PREDICT
7  ?project RESOURCE
8  ?topic RESOURCE
9  ?publication RESOURCE
10  ?personType RESOURCE
11  ?publicationType RESOURCE
12 }
13 WHERE {
14  ?person swrc:affiliation ?group .
15  ?person rdf:type ?personType .
16  OPTIONAL { ?person swrc:worksAtProject ?project } .
17  OPTIONAL { ?person swrc:worksAtTopic ?topic } .
18  OPTIONAL {
19   ?person swrc:publication ?publication .
20   ?publication rdf:type ?publicationType
21  }
22 }
23 USING <http://kdl.cs.umass.edu/proximity/rpt>
```

Listing 6.7: SPARQL-ML create query for the research group affiliation prediction.

The resulting subgraph instances created by SPARQL-ML are shown in Figure 6.10. We chose a different way to illustrate the subgraphs because of the increased amount of nodes in the subgraph.

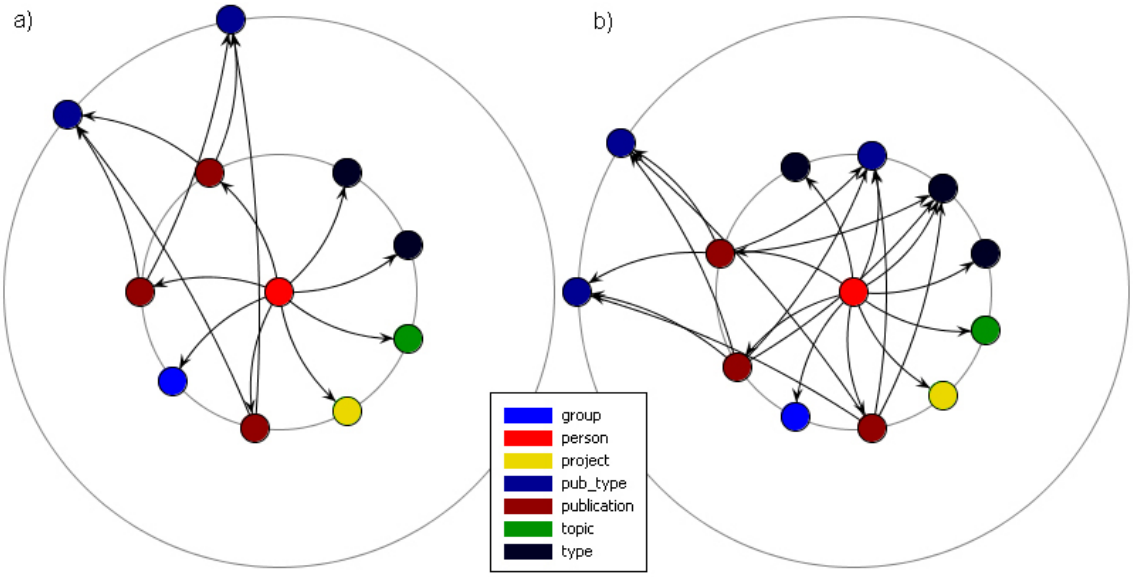


Figure 6.10: Example subgraphs built for the research group affiliation prediction, a) without inferencing and b) with inferencing.

6.4.3 Step 2: Prediction

The prediction query shown in Listing 6.8 outputs the persons, the actual research group they are affiliated with, and the prediction given by our classifier.

```
1  PREFIX sml:  <java:ch.uzh.ifi.sparqlml.mining.pfunction.>
2  PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
3  PREFIX swrc: <http://swrc.ontoware.org/ontology#>
4
5  SELECT DISTINCT ?person ?group ?prediction
6  WHERE {
7   ?person swrc:affiliation ?group .
8   ?person rdf:type ?personType .
9   OPTIONAL { ?person swrc:worksAtProject ?project } .
10  OPTIONAL { ?person swrc:worksAtTopic ?topic } .
11  OPTIONAL {
12   ?person swrc:publication ?publication .
13   ?publication rdf:type ?publicationType
14  } .
15  ?prediction
16          sml:predict ( <http://www.ifi.uzh.ch/ddis/affiliations>
17                        ?person ?group ?personType ?project ?topic
18                        ?publication ?publicationType )
19 }
```

Listing 6.8: SPARQL-ML predict query of the research group affiliation prediction.

6.4.4 Results and Discussion

The RBC clearly outperformed the RPT, hence, we concentrate only on the results given by the RBC. For both tasks, our tests with inferencing have shown that the additional inferred triples do not increase the prediction performance. Neither the types of a person nor the types of publications were good predictors for the research group affiliation.

Table 6.6 summarizes the macro-averaged results that were estimated via Leave-One-Out Cross-Validation (LOOCV) according to [Bloehdorn and Sure, 2007]. The first row represents the best results given by Bloehdorn and Sure with the same features, calculated with a Support Vector Machine and customized kernels for Semantic Web data. The second and third row show our results with an RBC with and without inferencing. As Table 6.6 shows, our method clearly outperforms the kernel-based approach in terms of prediction error, recall, and F-Measure (F_1), while having slightly lower precision. We argue that the inferior results of the RBC with inferred triples are due to a possible overfitting of the classifier.

The ease-of-use of our approach, which basically consists of writing a simple SPARQL-ML query, is a major benefit compared to the complex task of choosing from various kernels, kernel modifiers, and parameters.

person2affiliation						**publication2affiliation**				
algorithm	*err*	*prec*	*rec*	F_1		*algorithm*	*err*	*prec*	*rec*	F_1
sim-ctpp-pc, c=1	4.49	95.83	58.13	72.37		sim-cta-p, c=10	0.63	99.74	95.22	97.43
RBC w/o inf	3.53	87.09	80.52	83.68		RBC w/o inf	0.09	98.83	99.61	99.22
RBC w/ inf	3.67	85.72	80.18	82.86		RBC w/ inf	0.15	97.90	99.25	98.57

Table 6.6: LOOCV results for the *person2affiliation* and *publication2affiliation* tasks.

7

Limitations

In this chapter we discuss some of the major limitations of our approach and how they could be tackled to improve the use and performance of our implementation.

7.1 Conceptual Limitations

A major challenge was to find good datasets that can be used for data mining. To gain a good understanding of the data and to create models with reasonable support we are in need of complete and noise-free datasets. Most available datasets are not carefully selected nor up-to-date, hence, the task of predicting anything from this data will not yield good results. During this thesis, we came across a lot of datasets that were either incomplete or simply not expressive enough to allow an accurate prediction. Hence, we argue, that further experiments on data mining from Semantic Web data could be greatly facilitated with the creation of common datasets for the evaluation and comparison of different approaches.

The benefit of our approach is based on the expressiveness of the underlying ontologies. While ontologies with a deep inheritance hierarchy can outperform data mining without ontology support, the results for the prediction task are less likely to improve by applying an ontology with a flat hierarchy with less expressive concepts. Therefore, the careful and exact creation of ontologies is an important task to improve the results of data mining from Semantic Web data.

7.2 Technical Limitations

A technical limitation of our implementation is the inferior performance for the data mining task on large datasets. By exporting the data into the appropriate format for the import into a data mining module, we allow the use of a stand-alone knowledge discovery tool. However, this additional step takes more time than the direct import of the data into the mining module. To overcome this limitation, without sacrificing the benefit of an intermediate format, we could create a new parameter that allows the user to choose which technique he would like to use.

8

Conclusions and Future Work

This thesis presented a novel approach we call SPARQL-ML. It extends traditional SPARQL with data mining support to perform knowledge discovery in the Semantic Web. We showed how our framework enables to predict unseen data in a new dataset, based on the results of a mining model. In particular, we demonstrated how models trained by statistical relational learning methods (SRL) such as Relational Probability Trees (RPTs) and Relational Bayesian Classifiers (RBCs) outperform models not taking into account the additional information about the links between objects. Additionally, we improved the results of learning algorithms with the integration of implicit data, which can be made explicit through inferencing. We presented an implementation of our theoretical framework into ARQ/Jena, which was then used for our evaluation of SPARQL-ML.

Our approach is extensible in terms of the supported data mining algorithms, and generic as it is applicable for any Semantic Web datasets. We fully analyzed SPARQL-ML on synthetic and real-world datasets to show its excellent prediction and classification quality, as well as its superiority to other related approaches, such as kernel methods used in Support Vector Machines. Finally, given the usefulness and the ease of use of our novel approach, we believe that SPARQL-ML could serve as a standardized way on how to perform data mining tasks on Semantic Web data.

8.1 Future Work

Future work will investigate further enhancements of SPARQL-ML, such as the introduction of a `DELETE` query to allow the removal of learned models. The evaluations in this thesis concentrated mainly on Proximity as a mining module, but other relational learning tools like NetKit[1] or Alchemy[2] can also be integrated into our extension. This would ultimately help to easily compare different learning algorithms on the same dataset. A tighter implementation of Weka would also include the research of different propositionalization strategies for Semantic Web data. Instead of relying on existing data mining tools, a future goal is the design and integration of self-made

[1] http://www.research.rutgers.edu/sofmac/NetKit.html
[2] http://alchemy.cs.washington.edu/

Semantic Web learning algorithms which are influenced by the benefits of relational learning algorithms and the logic deduction capabilities offered by ontologies.

As another focus of our future research, we intend to investigate which features of an ontology could help to improve the performance of predictions. While our research has shown that a deep concept hierarchy is a reliable prediction factor, we have not yet empirically shown how the implicit information given by object or data properties could improve the data mining task. As another important field of future work, we can think of generating useful Semantic Web datasets for data mining purposes. This could lead to the creation of new datasets or the enhancement of exiting datasets with more expressive ontologies.

A

SPARQL-ML Grammar Extension

```
1 void Query() : { }
2 {
3   Prologue()
4   ( SelectQuery() | ConstructQuery() | DescribeQuery()
5   | AskQuery() | CreateQuery() )
6 }
```

Listing A.1: Grammar extension of the Query method to include the CreateQuery in the list of possible query types.

```
1 void CreateQuery() : { Node v ; Node miningModel; int type;
2                        int predict; ArrayList nomValues ;}
3 {
4  <CREATEMININGMODEL>
5   { getQuery().setQueryCreateType() ; }
6   (  miningModel = IRIref() {getQuery().setMiningModel(miningModel);}  )
7   <LBRACE>
8   (
9     (v = Var() { predict=0; type = -1; nomValues = new ArrayList(); }
10        (
11          (<RESOURCE> { type=0;} )
12          (<PREDICT> {predict=1;} | <TARGET> {predict=2;})?
13          (nomValues = NomList())?
14              { getQuery().addMiningVar(v, type, predict, nomValues); }
15          |
16          (<DISCRETE> { type=1; } )
17          (<PREDICT> {predict=1;} )?
18          (nomValues = NomList())?
19              { getQuery().addMiningVar(v, type, predict, nomValues); }
20          |
21          (<CONTINUOUS> { type=2; } )
22          (<PREDICT> {predict=1;} )?
23              { getQuery().addMiningVar(v, type, predict, nomValues); }
24        )
25    ) +
```

```
26      { getQuery().setQueryResultStar(false) ; }
27    )
28    <RBRACE>
29    ( DatasetClause() )*
30    WhereClause()
31    SolutionModifier()
32    UsingClause()
33  }
```

Listing A.2: The CreateQuery extension defines the grammar of our `CREATE MINING MODEL` query. It allows the specification of variables and their content types for the model learning process.

```
1  ArrayList NomList() : {Token t;
2                         ArrayList nomList = new ArrayList(); String value;}
3  {
4    (
5      <NIL>
6      |
7      <LBRACE>
8        t = <STRING_LITERAL1>
9          { value = stripQuotes(t.image) ; nomList.add(value) ; }
10       (<COMMA> t = <STRING_LITERAL1>
11         { value = stripQuotes(t.image) ; nomList.add(value) ; } )*
12     <RBRACE>
13   )
14   { return nomList ; }
15 }
```

Listing A.3: The NomList extension allows to specify a list of nominal values for a `DISCRETE` variable in the `CREATE MINING MODEL` query.

```
1  void UsingClause() : {Node miningAlg; ExprList args ;}
2  {
3    <USING> ({ miningAlg = IRIref(); }
4    { getQuery().setMiningAlg(miningAlg) ; })
5    ( args = ArgList() { getQuery().setMiningAlgArgs(args); } ) ?
6  }
7  }
```

Listing A.4: The UsingClause allows to specify the learning algorithm and its parameters.

B

Metadata of the Implemented Mining Modules and Algorithms

```
<smo:MiningApp rdf:about="http://www.cs.waikato.ac.nz/ml/weka">
 <smo:hasAppName>Weka</smo:hasAppName>
 <dc:creator>The University of Waikato</dc:creator>
 <smo:hasClass>ch.uzh.ifi.sparqlml.mining.module.weka.WekaApp</smo:hasClass>
</smo:MiningApp>

<smo:MiningApp rdf:about="http://kdl.cs.umass.edu/proximity">
 <smo:hasAppName>Proximity</smo:hasAppName>
 <dc:creator>University of Massachusetts Amherst -
 Knowledge Discovery Laboratory</dc:creator>
 <smo:hasClass>
  ch.uzh.ifi.sparqlml.mining.module.proximity.ProximityApp
 </smo:hasClass>
 <smo:hasConnection>localhost:30000</smo:hasConnection>
</smo:MiningApp>
```

Listing B.1: Metadata of the implemented mining modules (Weka and Proximiy) after the definitions in our SPARQL mining ontology (SMO).

```
<smo:Algorithm rdf:about="http://www.cs.waikato.ac.nz/ml/weka/j48">
 <smo:hasAlgorithmName>J48</smo:hasAlgorithmName>
 <smo:hasMiningApp rdf:resource="http://www.cs.waikato.ac.nz/ml/weka"/>
 <smo:hasClass>weka.classifiers.trees.j48</smo:hasClass>
</smo:Algorithm>

<smo:Algorithm rdf:about="http://www.cs.waikato.ac.nz/ml/weka/m5p">
 <smo:hasAlgorithmName>M5P</smo:hasAlgorithmName>
 <smo:hasMiningApp rdf:resource="http://www.cs.waikato.ac.nz/ml/weka"/>
 <smo:hasClass>weka.classifiers.trees.m5p</smo:hasClass>
</smo:Algorithm>

<smo:Algorithm
 rdf:about="http://www.cs.waikato.ac.nz/ml/weka/linearRegression">
 <smo:hasAlgorithmName>Linear Regression</smo:hasAlgorithmName>
 <smo:hasMiningApp rdf:resource="http://www.cs.waikato.ac.nz/ml/weka"/>
 <smo:hasClass>weka.classifiers.functions.LinearRegression</smo:hasClass>
</smo:Algorithm>

<smo:Algorithm rdf:about="http://kdl.cs.umass.edu/proximity/rbc">
 <smo:hasAlgorithmName>Relational Bayes Classifier</smo:hasAlgorithmName>
 <smo:hasMiningApp rdf:resource="http://kdl.cs.umass.edu/proximity"/>
 <smo:hasClass>kdl.prox3.model.classifiers.RBC</smo:hasClass>
</smo:Algorithm>

<smo:Algorithm rdf:about="http://kdl.cs.umass.edu/proximity/rpt">
 <smo:hasAlgorithmName>Relational Probability Tree</smo:hasAlgorithmName>
 <smo:hasMiningApp rdf:resource="http://kdl.cs.umass.edu/proximity"/>
 <smo:hasParam
  rdf:resource="http://kdl.cs.umass.edu/proximity/rpt#maxDepth"/>
 <smo:hasParam
  rdf:resource="http://kdl.cs.umass.edu/proximity/rpt#numThresh"/>
 <smo:hasParam
  rdf:resource="http://kdl.cs.umass.edu/proximity/rpt#pVal"/>
 <smo:hasClass>kdl.prox3.model.classifiers.RPT</smo:hasClass>
</smo:Algorithm>
```

Listing B.2: Metadata of the implemented algorithms after the definitions in our SPARQL mining ontology (SMO).

C

Accessing SPARQL-ML with Java

```
1 // Create new QueryML object with a CREATE MINING MODEL
2 // query string (QueryString).
3 QueryML query = QueryFactoryML.create(QueryString);
4
5 // Create query execution for the query object and the input data (DataModel).
6 QueryExecutionML qe = QueryExecutionFactoryML.create(query, DataModel);
7
8 // Execute create statement and receive the metadata of the learned model.
9 Model trainModel = qe.execCreate();
10
11 // Write the model to the output.
12 trainModel.write(System.out, "RDF/XML-ABBREV");
```

Listing C.1: Example on how to access the create statement of SPARQL-ML from Java source code.

```
1 // Create new QueryML object with the PREDICT query string (QueryString).
2 QueryML query = QueryFactoryML.create(QueryString);
3
4 // Create query execution for the query object and the input data (DataModel).
5 QueryExecutionML qe = QueryExecutionFactoryML.create(query, DataModel);
6
7 // Execute select query and receive the ResultSet.
8 ResultSet results = qe.execSelect();
9
10 // Write the results to the output.
11 ResultSetFormatter.out(System.out, results, query);
```

Listing C.2: Example on how to access the predict statement of SPARQL-ML from Java source code.

D

RPT of the Semantic Web Service Domain Prediction

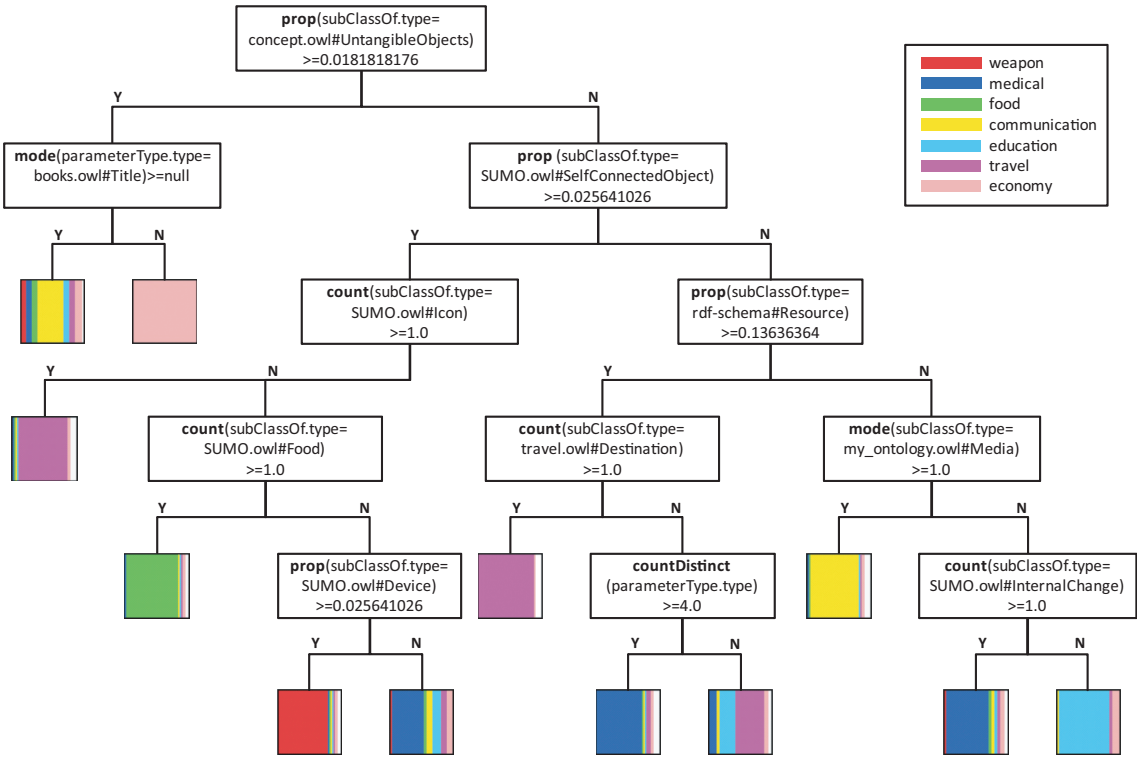


Figure D.1: RPT on inferred data for the Semantic Web service domain prediction.

E

CD

The enclosed CD-ROM contains all the data created during this thesis. Table E.1 describes the content of the different folders in more detail.

Folder	**Content**
Code	Contains the source code of SPARQL-ML and other helper programs used for data preprocessing.
Ontology	Contains the SPARQL Mining Ontology (SMO) with the necessary metadata files for SPARQL-ML.
Experiments	Contains the data used for our experiments in Chapter 6, as well as the detailed results and outputs of SPARQL-ML.
Text	Contains a digital version of this thesis and additional documentation.

Table E.1: Table of Contents of the enclosed CD-ROM.

List of Listings

List of Figures

List of Tables

Bibliography

[Anyanwu and Sheth, 2003] Anyanwu, K. and Sheth, A. (2003). ρ-Queries: Enabling Querying for Semantic Associations on the Semantic Web. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, pages 690–699, New York, NY, USA. ACM Press.

[Biron and Malhotra, 2004] Biron, P. V. and Malhotra, A., editors (2004). *XML Schema Part 2: Datatypes*. W3C Recommendation. W3C, second edition.

[Bloehdorn and Sure, 2007] Bloehdorn, S. and Sure, Y. (2007). Kernel Methods for Mining Instance Data in Ontologies. In Aberer, K., Choi, K.-S., and Noy, N., editors, *Proceedings of the 6th International Semantic Web Conference (ISWC 2007)*, Lecture Notes in Computer Science. Springer. to appear.

[Carroll et al., 2004] Carroll, J. J., Dickinson, I., Dollin, C., Reynolds, D., Seaborne, A., and Wilkinson, K. (2004). Jena: Implementing the Semantic Web Recommendations. In *WWW Alt. '04: Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, pages 74–83, New York, NY, USA. ACM.

[Chen et al., 2003] Chen, X., Zhou, X., Scherl, R. B., and Geller, J. (2003). Using an Interest Ontology for Improved Support in Rule Mining. In *DaWaK*, pages 320–329.

[Domingos and Provost, 2000] Domingos, P. and Provost, F. (2000). Well-trained PETs: Improving Probability Estimation Trees.

[Dzeroski, 2003] Dzeroski, S. (2003). Multi-Relational Data Mining: an Introduction. *SIGKDD Explorations*, 5(1):1–16.

[Edwards et al., 2002] Edwards, P., Grimnes, G., and Preece, A. (2002). An Empirical Investigation of Learning from the Semantic Web.

[Fayyad et al., 1996] Fayyad, U. M., Piatetsky-Shapiro, G., and Smyth, P. (1996). From Data Mining to Knowledge Discovery: An Overview.

[Getoor et al., 1999] Getoor, L., Friedman, N., Koller, D., and Pfeffer, A. (1999). Learning Probabilistic Relational Models. In *IJCAI*, pages 1300–1309.

[Getoor and Licamele, 2005] Getoor, L. and Licamele, L. (2005). Link Mining for the Semantic Web, Position Statement. In *Proceedings of the Dagstuhl Seminar in Machine Learning for the Semantic Web*.

[Getoor and Taskar, 2007] Getoor, L. and Taskar, B. (2007). *Introduction to Statistical Relational Learning (Adaptive Computation and Machine Learning)*. The MIT Press.

[Gilardoni et al., 2005] Gilardoni, L., Biasuzzi, C., Ferraro, M., Fonti, R., and Slavazza, P. (2005). Machine Learning for the Semantic Web: Putting the User into the Cylce. In *Proceedings of the Dagstuhl Seminar in Machine Learning for the Semantic Web*.

[Hartmann, 2004] Hartmann, J. (2004). A Knowledge Discovery Workbench for the Semantic Web. In *Workshop on Mining for and from the Semantic Web at the ACM SIGKDD*.

[Hess and Kushmerick, 2004] Hess, A. and Kushmerick, N. (2004). Machine Learning for Annotating Semantic Web Services. In *AAAI Spring Symposium on Semantic Web Services*.

[Jiang and Tan, 2006] Jiang, T. and Tan, A.-H. (2006). Mining RDF Metadata for Generalized Association Rules: Knowledge Discovery in the Semantic Web Era. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, pages 951–952, New York, NY, USA. ACM Press.

[Kiefer et al., 2007] Kiefer, C., Bernstein, A., and Stocker, M. (2007). The Fundamentals of iSPARQL - A Virtual Triple Approach For Similarity-Based Semantic Web Tasks. In *Proceedings of the 6th International Semantic Web Conference (ISWC)*, Lecture Notes in Computer Science, pages 295–309. Springer.

[Kochut and Janik, 2007] Kochut, K. and Janik, M. (2007). SPARQLeR: Extended Sparql for Semantic Association Discovery. In Franconi, E., Kifer, M., and May, W., editors, *ESWC*, volume 4519 of *Lecture Notes in Computer Science*, pages 145–159. Springer.

[Liu, 2007] Liu, B. (2007). *Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data*. Data-centric Systems and Applications. Springer.

[Martin et al., 2004] Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Parsia, B., Payne, T., Sirin, E., Srinivasan, N., and Sycara, K. (2004). OWL-S: Semantic Markup for Web Services.

[Mcguinness and van Harmelen, 2004] Mcguinness, D. L. and van Harmelen, F. (2004). OWL Web Ontology Language Overview. W3C recommendation, W3C.

[Neville and Jensen, 2004] Neville, J. and Jensen, D. (2004). Dependency Networks for Relational Data. In *ICDM '04: Proceedings of the Fourth IEEE International Conference on Data Mining (ICDM'04)*, pages 170–177, Washington, DC, USA. IEEE Computer Society.

[Neville et al., 2003a] Neville, J., Jensen, D., Friedland, L., and Hay, M. (2003a). Learning Relational Probability Trees. In Getoor, L., Senator, T. E., Domingos, P., and Faloutsos, C., editors, *KDD*, pages 625–630. ACM.

[Neville et al., 2003b] Neville, J., Jensen, D., and Gallagher, B. (2003b). Simple Estimators for Relational Bayesian Classifiers. In *ICDM*, pages 609–612. IEEE Computer Society.

[Neville et al., 2003c] Neville, J., Rattigan, M., and Jensen, D. (2003c). Statistical Relational Learning: Four Claims and a Survey. In *Proceedings of the Workshop on Learning Statistical Models from Relational Data, Eighteenth International Joint Conference on Artificial Intelligence*.

[Pease et al., 2002] Pease, A., Niles, I., and Li, J. (2002). The Suggested Upper Merged Ontology: A Large Ontology for the Semantic Web and its Applications. In *Working Notes of the AAAI-2002 Workshop on Ontologies and the SemanticWeb*, Edmonton, Canada.

[Perez et al., 2006] Perez, J., Arenas, M., and Gutierrez, C. (2006). The Semantics and Complexity of SPARQL.

[Provost and Fawcett, 2001] Provost, F. J. and Fawcett, T. (2001). Robust Classification for Imprecise Environments. *Machine Learning*, 42(3):203–231.

[Prud'hommeaux and Seaborne, 2007] Prud'hommeaux, E. and Seaborne, A. (2007). SPARQL Query Language for RDF. Technical report, W3C. W3C Proposed Recommendation 12 November 2007.

[Sabou, 2005] Sabou, M. (2005). Learning Web Service Ontologies: Challenges, Achievements and Opportunities. In *Proceedings of the Dagstuhl Seminar in Machine Learning for the Semantic Web*.

[Valiente, 2002] Valiente, G. (2002). *Algorithms on Trees and Graphs*. Springer-Verlag, Berlin.