Karlsruhe Institute of Technology

Automatic & Semi-Automatic Methods for Supporting Ontology Change

Zur Erlangung des akademischen Grades eines Doktors der
Wirtschaftswissenschaften (Dr. rer. pol)
von der Fakultät für Wirtschaftswissenschaften
des Karlsruher Institut für Technologie (KIT)

Genehmigte Dissertation
von

Dipl.-Inform. Uta Lösch

Referent:	Prof. Dr. Rudi Studer
Korreferent:	Prof. Dr. Andreas Hotho
Tag der mündlichen Prüfung:	09.02.2012

Abstract

The Semantic Web vision consists in encoding the knowledge made available on the web in a way which can be understood by machines as well as by humans. Over the last years more and more semantic data has been published on the web. Semantic data is made available in the form of ontologies – formal models of a domain of interest. As the underlying domain or the requirements for a specific ontology may change, the knowledge encoded in it is also changing over time. Therefore, changes to the ontologies are necessary. However, these changes are complicated, as the knowledge in the ontology is usually highly interlinked and only experts may be able to understand the complete formalization.

The main topic of this thesis is the development of methods which support and thus facilitate the ontology change process.

The contributions of this thesis are threefold: Firstly, existing change support methods are analysed and classified according to a newly proposed classification scheme.

Secondly, a method for inducing additional facts from the existing ontology is proposed. Our contribution lies in the definition of kernel functions for Resource Description Framework (RDF) data which may be used for classifying entities as well as for predicting links between entities. The proposed methods may be used for completing an existing ontology with facts that have not explicitely been stated and are not derivable (or rejectable) by means of deductive methods.

Thirdly, a framework is proposed which allows for the automatic handling of complex update requests. The approach is based on the identification of change patterns, which describe sets of frequently occurring changes. Such changes may be predefined in the proposed framework and then be instantiated later on. This approach allows users who are not familiar with the precise formalization of knowledge in the ontology to perform changes on it.

Contents

Part I

Introduction

Chapter 1

Introduction

With the broad uptake of the Semantic Web more and more semantic data is made available on the web. As an official standard of the World Wide Web Consortium (W3C), the *Resource Description Framework (RDF)* establishes a universal graph-based data model which is sometimes claimed to form the backbone of the so-called "Semantic Web" (Manola and Miller, 2004). Recent efforts of research, industry and public institutions in the context of Linked Open Data (LOD) initiatives have led to considerable amounts of RDF data sets being made available and linked to each other on the web (Bizer et al., 2009a). Semantic Web data is available nowadays for many domains: wikipedia facts[1], government data[2], geo data[3] etc. As an illustration of the size of today's Semantic Web, consider the statistics of the search engine SINDICE[4], which in June 2011 counts a total of over 260 million indexed RDF documents.

Semantic data is available on the Semantic Web in the form of so-called ontologies which provide descriptions of concepts, relations and individuals in a domain of interest. The manual creation of these ontologies is expensive and difficult. Thus, various ways for supporting the process of making data available have been proposed: from support of the manual generation to extraction from text or wrappers around "traditional" data sources like relational database systems.

No matter which possibility was used for creating a data source, the problem of its maintenance arises. New information about the domain may become available, information about the described entities may have to be updated or modelling errors may be detected. The issue of incorporating new information is particularly complex in the context of Semantic Web data:

- In these data sources the information is often not represented explicitely in

[1] `http://dbpedia.org`
[2] `http://data.gov.uk/linked-data`
[3] `http://ckan.net/dataset/linkedgeodata`
[4] `http://sindice.com`

the knowledge base. The logic formalisms underlying Semantic Web data representations define reasoning methods which allow for the deduction of additional knowledge from the ontology by means of logic inference. This inferred knowledge is difficult to handle in the context of update and change operations.

- The limited expressivity of lightweight ontology data limits the possibility of constraining relations between entities of interest. However, in the modeled data such constraints may exist, e.g. relations or type memberships that can not occur together.

- Specific changes to an ontology may require additional changes in order to obtain a complete, consistent, coherent ontology which does not contain undesired information. An example of this difficulty is the deletion of data which is required for a specific entity. In this case, the deleted information has to be replaced by some new updated information.

A second problem is that often information is missing in the knowledge base as it cannot be stated with certainty as would be required by the logical formalisms underlying Semantic Web data or as it was just forgot during the modelling phase. Reasoning approaches define rules for deducing facts that have to be true based on the given ontology. However, these rules are often not helpful for searching the precise facts that hold in special cases, e.g. it may be possible to infer from an ontology that an entity has to have a specific property, but a statement about its value may not be possible. As a concrete example, consider an ontology where it may be deduced that a person has to have a gender, but the gender of a concrete person may not be deducable. Inductive – data-driven – approaches can deal with these issues which can not be overcome using classical reasoning regimes: The analysis of the data with statistical methods may be able to derive knowledge which is not made explicit in the knowledge base but which still has a high likelihood of being true based on the explicit knowledge.

The complexity of ontology maintenance makes methods for supporting ontology change desirable, which support the ontology engineer in incorporating new information into the ontology, in detecting problems of the current ontology state and in adapting an ontology for new applications. In this thesis, the ontology support process is analysed, an overview of existing ontology change support methods is given and the problems of current support methods are analysed. Based on this analysis, two research questions are identified and the solutions for these problems which are developed in the scope of this thesis are presented. In the following, an overview of the research questions and the proposed solutions is given (see Section 1.1), before presenting an overview of this thesis in Section 1.2.

1.1 Research questions and Contributions

The problems described above are addressed in this thesis through the following research questions:

- *How can uncertain facts be induced from a lightweight ontologies in a domain-independent way?*

  This research question has been addressed in two parts: we have developed kernel-based methods for classifying entities in an RDF dataset and for link prediction in RDF datasets (Lösch et al., 2012).

  The usage of RDF data in machine learning tasks requires the definition of suitable data representations that allow for the integration of these new kinds of data into existing machine learning algorithms. We have developed a set of kernel functions for learning from RDF data based on the underlying graph representation. In our experiments we can show that the proposed kernel functions achieve competitive results with handcrafted semantics-based data representations and can outperform classical graph kernels applied to RDF graphs.

  In a second part we have developed a kernel-based approach for link prediction in RDF data and have instantiated it with the kernel functions for RDF entities. In our evaluation we show that the proposed link prediction method combined with our kernel functions can outperform a statistical relational learning approach.

- *How can knowledge bases be updated automatically, such that new information can automatically be incorporated into the knowledge base without making it inconsistent or incoherent?*

  The second research question has been addressed by developing a method for automatically processing frequently occurring changes in an ontology (Lösch et al., 2009). The approach consists in having the knowledge engineer define how certain changes are to be dealt with by the knowledge base beforehand and to carry out these additional changes when a change request is submitted. A system architecture and its instantiation for RDF knowledge bases are developed. The approach has been extended to an interactive setting where the knowledge worker is supported in the change process based on pre-defined change definitions and the actual changes to be made to the ontology are obtained from the interaction of the knowledge worker and the system.

1.2 Overview of this thesis

In Chapters 2 and 3 we will introduce the foundations of Semantic Technologies and Machine Learning, the two fields of research in which the contributions of this thesis are situated. We will define the most important terms and give definitions of the methods and terms that will be used later on in the thesis. In Chapter 2 we will present foundations of the Resource Description Framework (RDF), the Simple Protocol and RDF Query Language (SPARQL) and SPARQL Update. Chapter 3 presents background on kernel methods, Support Vector Machines (SVMs) and graph mining.

Then, the research carried out in the scope of this thesis is presented: Chapter 4 presents a classification of ontology changes and the methods proposed in earlier work to support the ontology change process. We will present a classification schema for the support methods which is based on an analysis of the reason for changing the ontology and the phase of the change process which is supported by this work. We will discuss existing approaches and their classification into the proposed classification scheme. Based on this analysis, we will identify issues which are not sufficiently addressed by existing approaches and motivate our research questions from them.

Chapter 5 and 6 present a solution for learning in light-weight (RDF) ontologies. In Chapter 5 a set of kernel functions for RDF entities based on intersection graphs resp. intersection trees are presented. The proposed kernel functions compare RDF entities based on common elements in the neighborhood of the entities in the data graph. Theoretical results are presented which identify structures in the intersection graph and the intersection tree based on which valid kernel functions are defined. The proposed kernel functions are evaluated and compared to general graph kernels and other kernel functions which have been deviced specifically for Semantic Web data in two evaluation scenarios: prediction of affiliations in the SWRC dataset, a dataset representing a research group with projects and publications, and prediction of a user's age in social network data. While in Chapter 5 the focus lies on the problem of classifying RDF entities, we show how the proposed kernel functions can be used for link prediction in RDF data sets. Therefore, a kernel function for RDF links is defined based on the kernel functions presented in Chapter 5. The adapted method is evaluated in two scenarios and compared to statistical link prediction methods.

Chapter 7 presents an approach for supporting changes in ontology update scenarios: a framework for dealing with "expected" ontology changes, i.e. to deal with kinds of changes that frequently occur, is presented. The approach allows for the specification of procedures to deal with these changes. The design choices that had to be made when designing this system, are presented and discussed. A language for defining these specifications in the case of RDF knowledge bases

is defined. In a number of examples the usefulness and the flexibility of the approach are demonstrated. An interactive extension of the basic approach is also presented in this chapter.

In Chapter 8 the results of this thesis are summarized and an outlook to future applications and research is given.

Part II

Foundations

Chapter 2

Semantic Technologies

The idea behind the development of Semantic Technologies is to encode the meaning and the representation of data separately, such that machines and humans can understand and work with the data. Thus, the ultimate goal is to make data understandable to machines.

While these ideas have been studied for decades in the context of knowledge representation in Artificial Intelligence, they have only lately been applied in the context of the World Wide Web.

The so-called *Semantic Web* aims at bringing semantic data to the web, i.e. to allow machines to understand the meaning of data that is published on the Web (Berners-Lee et al., 2001). In the past ten years a lot of effort has been put into achieving this vision.

An important concept for achieving the Semantic Web vision is that of an ontology:

Definition 1 (Ontology) *An* ontology *is a formal, explicit specification of a shared conceptualisation (Studer et al., 1998).*

An ontology thus is a description of a domain of interest in terms of concepts that are relevant for this domain. These relevant concepts and constraints on these concepts have to be made explicit. The whole ontology has to be declared using a formalism which is machine-readable (i.e. not in natural language). Last but not least the ontology is a "shared" conceptualisation, which means that a group of people has agreed on this representation of the knowledge about the domain of interest.

Ontologies constitute one of the central concepts in the field of Semantic Web technologies and facilitate information integration and exchange as well as semantic search. They are perceived as the building blocks of the Semantic Web. Usually, the expressive power of ontologies exceeds that of traditional databases and allows to infer new information that is not explicitly present in the specification but a logical consequence of it (the so-called *implicit knowledge*).

Typically, an ontology consists of two parts: *terminological knowledge*, which defines the concepts, properties and relations which exist in the domain, and *assertional knowledge* which defines concrete instances of the concepts and their relations among each other. The World Wide Web Consortium (W3C) has developed and published standards for defining various kind of data: Facts and relations may be expressed using the Resource Description Framework (RDF), the Resource Description Framework Schema (RDFS) is used to express type relations as well as domain and range restrictions, more complex restrictions (like disjointness of classes) can be stated using the Web Ontology Language (OWL). Finally, semantic data may be queried using the Simple Protocol and RDF Query Language (SPARQL), and it may be updated using SPARQL Update.

In the following sections, those technologies that are of importance in the context of this thesis, namely RDF, RDFS, SPARQL and SPARQL/Update will be presented.

2.1 Resource Description Framework (RDF)

The *Resource Description Framework (RDF)* (Manola and Miller, 2004) standard, which provides a versatile graph-based data model, connecting resources and basic data values by typed links, forms the backbone of the current Semantic Web. RDF is the data representation for which all the methods proposed in this thesis were conceived and implemented. In Chapter 5 we will develop methods for learning from RDF data. In Chapter 6 these methods will be applied to the problem of link prediction in RDF. Finally, the implementation provided for the ontology update framework presented in Chapter 7 enables automatic updates of RDF knowledge bases.

2.1.1 RDF data model

RDF is based on the idea of describing and linking arbitrary entities (identified by Uniform Resource Identifiers (URIs)) and data values by means of typed links (also identified by URIs) in sets of simple *statements* ("RDF triples").

The structure of the statement triples is devised in analogy with the basic structures we find in natural language sentences. Correspondingly, the first argument of each statement (any entity name) is referred to as the *subject*, the second argument (any property name) as the *predicate*, and the third argument (any entity name or value of one of the admitted data types) as the *object*. Hereby, RDF uses URIs identifying the subjects and predicates in statements. In case the object of a triple is an entity, it is also identified by a URI, in case the predicate denotes a data-typed relation, the object is formed by the datatype value.

Example 2 (People and Topics) *RDF allows us to encode basic information about persons as follows (RDF abstract syntax)*[1]*:*

```
person100     foaf:name            "Uta Lï¿½sch"
person100     foaf:topic_interest  topic110
topic110      skos:prefLabel       "Machine Learning"
person100     foaf:knows           person200
person200     foaf:name            "Achim Rettinger"
person200     foaf:topic_interest  topic110
person100     foaf:topic_interest  topic120
topic120      skos:prefLabel       "Ontology Update"
person100     foaf:knows           person300
person300     foaf:name            "Sebastian Rudolph"
person300     foaf:topic_interest  topic120
```

Blank nodes are a special type of nodes in RDF. This kind of node is e.g. used for modelling n-ary relations. A blank node is an RDF node just like any other, but it does not have a URI. It is thus not possible to reference this node - the node only gets an identifier for the purpose of serializing the graph. RDF additionally includes some more advanced concepts like the specification of sets and lists (see (Manola and Miller, 2004) for more information on these advanced concepts) or the possibility to make statements about statements (so-called *reification*). All these complex data structures are again represented by means of triples.

The triples represent a graph structure, where each entity name resp. datatype value which occurs as either subject or object of a triple represents a node in the graph. Each triple describes an edge in the graph: the edge links the subject to the object of the triple, the predicate defines the label of the edge. A set of RDF statements thus implicitly describes a *directed* and *labeled graph*. The structure and labels of the graph represent the overall RDF knowledge structure.

Example 3 (People and Topics cont.) *The graph corresponding to the triples in our example is shown in Fig. 2.1.*

Over the last years a lot of effort has been put into making RDF data available on the web. Especially the effort around Linked Open Data (LOD) has got a lot of attention. The people involved in this initiative are pushing the publication of RDF datasets on the web in a way that these datasets are interlinked.

[1]The statements use property names taken from well-known RDF-based metadata standards such as Friend of a Friend (FOAF) (Brickley and Miller, 2007) and Simple Knowledge Organisation Systems (SKOS) (Miles and Brickley, 2005), marked by the corresponding foaf and skos namespaces.

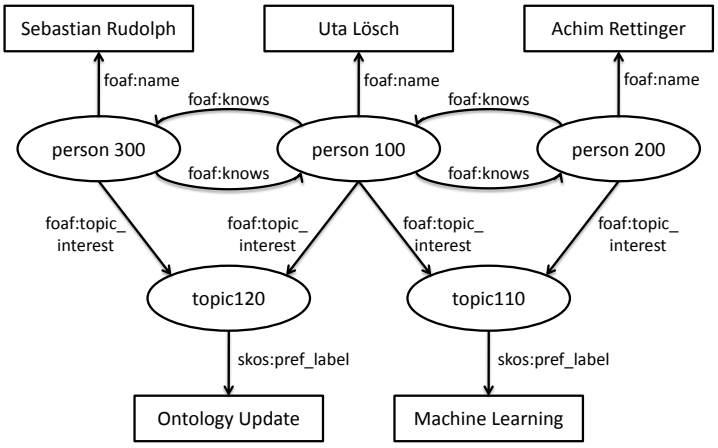

Figure 2.1: RDF graph corresponding to triples in Example 2

The most well-known data set which was published as part of the LOD effort, is DBpedia, which is a dataset representing Wikipedia's entities, categories as well as some data which can automatically be extracted from Wikipedia pages (Bizer et al., 2009b).

2.1.2 Resource Description Framework Schema (RDFS)

With its extension RDFS, RDF constitutes a so-called "leightweight" ontology language, providing basic modeling features for assertional (instance-related) and terminological knowledge handling classes, binary relations (so-called properties), hierarchies of classes (also referred to as taxonomies) and properties as well as domain and range specifications for properties.

Specifically, RDFS is a vocabulary with specific properties for modelling meta-information on the presented data, i.e. schema information for the data. The most important among these properties are `rdfs:label` for giving human-readable labels to URIs, `rdfs:domain` and `rdfs:range` for denoting the domain and the range of a property, and `rdfs:subClassOf` for denoting subsumptions between two concepts.

Example 4 (People and Topics cont.) *RDFS allows for the encoding of restrictions on the properties and the concepts in the data graph. In the example we express that the* `foaf:name` *property links a person to a literal:*

```
foaf:name     rdfs:domain          foaf:Person
foaf:name     rdfs:range           rdfs:Literal
...
```

The notation of RDFS triples is the same as for RDF triples. However, their semantics is a different one as we will see in the following section.

2.1.3 RDF Semantics

RDF offers formal semantics which are defined in the RDF specification (Hayes, 2004). These semantics allow for basic reasoning over RDF data. While only few operations are possible on RDF itself, it becomes possible to infer entity types via domain and range restrictions of properties as well via type hierarchies using RDFS. However, the reasoning possibilities within RDF/RDFS remain limited, as complex restrictions like intersection or disjointness can not be expressed using these formalisms. For more complex modelling capabilities it would be necessary to resort to OWL (Hitzler et al., 2009).

Definition 5 (Interpretation set) *A simple interpretation $\mathcal{I}$ of an RDF vocabulary V is defined by:*

- *A non-empty set IR of resources, called the domain or universe of $\mathcal{I}$.*
- *A set IP, called the set of properties of $\mathcal{I}$.*
- *A mapping I_{EXT} from IP into the powerset of $IR \times IR$ i.e. the set of sets of pairs (x, y) with x and y in IR .*
- *A mapping I_S from URI references in V into $(IR \cup IP)$*
- *A mapping I_L from typed literals in V into IR.*
- *A distinguished subset LV of IR, called the set of literal values, which contains all the plain literals in V*

Based on the above sets and functions an interpretation function is defined:

Definition 6 (Interpretation function) *The interpretation function $.^{\mathcal{I}}$ is defined as:*

- *Each untyped literal ■a■ is mapped to a.*
- *Each typed literal l is mapped to $I_L(l)$: $l^{\mathcal{I}} = I_L(l)$*
- *Each URI u is mapped to $I_S(u)$: $u^{\mathcal{I}} = I_S(u)$*

This means that untyped literals are mapped to themselves. There are no strict requirements for the definition of typed literals.

The basic interpretation function is then extended such that a truth value is assigned to each *grounded triple*, i.e to each triple which does not contain any variables:

Definition 7 (Interpretation of Grounded triples) *The interpretation of grounded triples is defined by*

$$s\ p\ o.^{\mathcal{I}} = \begin{cases} true & s, p, o \in V \wedge (s^{\mathcal{I}}, o^{\mathcal{I}}) \in I_{EXT}(p^{\mathcal{I}}) \\ false & else \end{cases}$$

Thus, a grounded triple is true if its elements are in the vocabulary and if the pair of resources denoted by `s` and `o` is in the extension of the property represented by `p`. Based on the definition of the truth value of triples, the truth value of a graph can be determined. The interpretation of a graph returns *true*, if and only if the interpretation of all triples within the graph is *true*.

The last part of the simple interpretation is the interpretation of blank nodes, i.e. of nodes which are not denoted by a specific label. Basically, a graph containing blank nodes is *true* if for each blank node there exists a resource with which the node can be identified.

Definition 8 (Interpretation of blank nodes) *Let A be a function which associates all blank nodes with a resource from IR. Based on this function and the interpretation function $I + A$ the combined interpretation $\mathcal{I} + A$ is defined:*

$$x^{\mathcal{I}} + A = \begin{cases} A(x) & \text{if } x \text{ is a blank node} \\ x^{\mathcal{I}} & \text{else} \end{cases}$$

The interpretation of the RDF and RDFS vocabularies is then obtained as an extension of this basic interpretation. In order for an interpretation of the graph to be an RDF interpretation the basic triples from the RDF vocabulary (such as `rdf:type rdf:type rdf:property`) have to be true.

For the RDFS interpretation, an additional set describing the set of classes and an interpretation function for the class membership is introduced. Additionally, rules are introduced for the interpretation of the elements of the RDFS vocabulary (such as `rdfs:domain` and `range`). Again, a set of additional triples is introduced which have to be fulfilled in order for the interpretation to be an RDFS interpretation.

2.2 SPARQL

For querying RDF data, the W3C standard Simple Protocol and RDF Query Language (SPARQL) is available (Prud'hommeaux and Seaborne, 15 Jan. 2008). SPARQL is similar in spirit to SQL for databases, i.e. it allows to query an RDF knowledge base for data which has certain properties. SPARQL graph patterns are used in Chapter 7 for defining update patterns.

The key concept of constructing queries in SPARQL is the specification of graph patterns, i.e. graphs which contain variables as placeholders for node or edge labels. These graph patterns are then matched on the data graph. Each possible matching of the graph pattern on the data graph yields a variable binding. Using these variable bindings it is possible to retrieve data with certain properties. SPARQL allows to display the retrieved data directly or to construct new graphs based on it.

Graph patterns are formed by a set of triples, where triple elements may be replaced by variables (distinguished by a question mark before the identifier). Additionally to the query pattern, it has to be specified what should be done with the results of the query. The most important possibility is to show the bindings of specific variables in the matchings of the graph pattern (SELECT query).

Definition 9 (Basic Graph Pattern) *A Basic Graph Pattern is a set of triple patterns.*

A triple pattern is an element of the set: $(RDF\text{-}T \cup V) \times (R \cup V) \times (RDF\text{-}T \cup V)$ *where R denotes the set of all resources denoted by a URI, RDF-T denotes all elements of the RDF vocabulary, i.e. the set of named resources, blank nodes and literals and V a set of variables which is disjoint from RDF-T.*

The selection criteria for data are specified using basic graph patterns (this happens in the `where`-clause of the query. The basic graph pattern is matched against the data graph. Hereby, the interest lies in finding those elements which the defined variables stand for. Thus, each variable may stand for any node in the graph, the other elements in the graph pattern are matched to nodes having the same identifier. This graph matching yields a set of variable bindings which are then used for the result presentation. More complex conditions for the graph matchings can be specified: it is possible to specify parts which are matched optionally or to intersect (or union) the sets of variable bindings obtained from matching different graph patterns.

The other parts of the query specify what should be done with the matches that are found using the search criteria in the basic graph pattern. There are several kinds of queries which specify what should be done with the variable bindings found using the basic graph patterns. Probably the most interesting kind of queries are `select`-queries, which simply present the values the variables in the `select`-clause may stand for in the graph.

Example 10 (People and Topics cont.) *In our example, we query for people who are interested in the topic with label "Machine Learning".*

```
SELECT ?personName
WHERE { ?person foaf:name           ?personName.
        ?person foaf:topic_interest ?topic.
        ?topic  skos:prefLabel      "Machine Learning".}
```

The graph pattern describes a query for entities which have a `foaf:name` *link to another node in the graph. The subject of the first triple should also have a link of type* `foaf:topic_interest` *to some node which itself has a link of type* `skos:prefLabel` *to the datavalue node with label "Machine Learning". More intuitively, the query retrieves the name of people who are interested in a topic*

with label "Machine Learning". The result of this query is a table containing all possible bindings of the variable `?personName` *in matchings of the graph pattern on the data graph:*

`?personName`
"Uta Lösch"
"Achim Rettinger"

Additional query types are `construct`, `ask` and `describe` queries. The first type of query allows for the construction of graphs based on templates specified in the `construct` clause which are then filled with the variable bindings obtained in the `where` clause. The `ask` query checks whether any match of the `where` clause in the data graph exists. Finally, the `describe` query is used to obtain descriptions of the selected resources without knowing what a description looks like. This is useful in cases where the structure of the data set is not or only partially known and it is not clear which properties of the resources are of interest.

SPARQL endpoints serve as a means to make (RDF) knowledge bases accessible to humans and machines.[2] Besides the definition of a query language for RDF, the W3C recommendation for SPARQL also contains the definition of a protocol for the communication between the SPARQL enpoint and the machine/human querying the data (Clark et al., 2008).

2.3 SPARQL Update

SPARQL Update (Schenk et al., 2008) is an extension of the SPARQL standard which allows for updating and changing the data in an RDF knowledge base via the SPARQL protocol. Similar to the update part of SQL, SPARQL Update offers functionality for adding and deleting data in a knowledge base as well as the possibility to change data. The change language proposed in 7 is an extension and adaptation of SPARQL Update.

The basic operations are `insert` and `delete` which allow the direct insertion or deletion of a set of triples. The set of triples to be changed is specified using a basic graph pattern (as is used in the `where`-clause of the SPARQL queries. It is also possible to specify the set of triples to be changed directly using the commands `insert data` resp. `delete data`. There are two variants of the delete operation: either a graph pattern is specified based on which triples which are to be deleted are constructed, or all triples which match the graph pattern are deleted directly. It is furthermore possible to combine `delete` and `insert`

[2]Note that non-RDF data can also be made accesible through a SPARQL endpoint: the results obtained for the queries have to be described in the expected format and the obtained results have to be coherent with the RDF semantics.

operations in a so-called `modify` operation, this makes it for example possible to change properties of certain entities.

Example 11 (Bibliographic Metadata and Topic Hierarchies cont.) *To continue our example, imagine that we would like to add a new group - the Machine Learning Special Interest Group (ML-SIG). All people interested in Machine Learning will be members of this group:*

```
insert{ ML-SIG  foaf:member    ?person.}
where{  ?person foaf:interest  ?topic.
        ?topic  skos:prefLabel "Machine Learning".}
```

This will add the triples `ML-SIG foaf:member person100.` *and* `ML-SIG foaf:member person200.` *to the knowledge base.*

Additionally, the creation and deletion of graphs is possible through SPARQL Update operations.

Chapter 3

Machine Learning

Machine Learning is concerned with building systems that are able to improve their performance with experience (Mitchell, 1997). Many different kinds of problems can be solved using learning systems, e.g. chess playing, movie recommendation, news categorization, etc.

Formally, Mitchell (1997) defined a learning system as:

Definition 12 (Learning System) *A computer program is said to* learn *from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P improves with experience E.*

Machine learning is a very broad field and only a very limited subset of the methods which were developed in the Machine Learning community is relevant for this thesis. All methods developed here contribute to the field of *Kernel Methods*. We are particularly interested in kernel methods for graph data.

In the subsequent sections, we will first present some basic Machine Learning terms. We will then focus on kernel methods and SVMs as the best-known kernel-based classifiers. The subsequent section 3.3 will deal with mining graph data (focusing on graph classification) before we conclude with the presentation of the evaluation methodology for classification models. The interested reader is referred to (Mitchell, 1997) or (Tan et al., 2006) for additional information regarding Machine Learning and the most widely used algorithms in this field. For more information on Kernel Methods you may refer to (Shawe-Taylor and Christianini, 2004).

3.1 Machine Learning Foundations

Learning systems are based on observations or experiences that improve the system performance. These experiences are made available to the machine learning algorithm as *input data*.

Each learning problem then requires:

- The choice of a suitable data representation, i.e. a suitable data model and a suitable description of each element from the input data in the data model
- The choice of a suitable class of hypotheses $\mathcal{H}$ (the *model class*).
- A method for choosing a specific model function $f \in \mathcal{H}$ by adjusting the free parameters of functions within $\mathcal{H}$.

For example, consider a scenario where texts are to be classified into various topic categories. A possible data representation is vectors with terms as attributes and TFIDF as feature weights, a class of hypotheses is decision trees and one of the algorithms that can be used to choose a specific decision tree is the C4.5 algorithm.

The choice of a suitable data representation is not only influenced by the problem at hand and the input data which is available, but also by the model class which is chosen and the algorithms which are available for choosing a specific model. Most learning algorithms' input data consists of a vector describing each experience. Here, we will concentrate on learning based on vectorial input data; Machine Learning (ML) algorithms for other kinds of input data, especially graphs, as well as methods for making these alternative kinds of input data usable for classical ML algorithms will be discussed in Sections 3.3.

An experience in the input data is called *instance* and is denoted by a vector containing the value of the data object for each of its *attributes*. Each instance can thus be described as a vector $x_i = (x_{i1}, \ldots, x_{im})$ and the whole dataset can be formalized as a matrix $D = (x_1, \ldots, x_n)^\top$. The colums in the data matrix are called *attributes* or *features* and are here denoted by $a_1, \ldots, a_m$.

Attributes are distinguished depending on the values which they can take: *Nominal attributes* are attributes which can only be tested for equality, while for *numeric attributes* an order on the values can be established and values can be compared. A typical nominal attribute is an attribute denoting a colour, a typical example for a numeric attribute is the age of a person.

Learning problems are generally divided in two classes: *supervised learning problems* and *unsupervised learning problems*. In the following, the two problem classes will be introduced.

3.1.1 Supervised Learning

Supervised learning is used for solving so-called *prediction tasks*. In these tasks, each instance consists of two parts: a *target value* y_i and the *input instances* x_i which are defined by input values x_{ij}. The learning problem then consists in finding a function f, such that (in the ideal case) $f(x_i) = y_i$ for all instances.

The two main problems within this class are *classification* (also known as *categorization*) and *regression*. The difference between the two problem classes is that in the case of regression the predicted variable y_i is a numerical variable and in categorization problems it is a nominal variable.

3.1.2 Unsupervised Learning

Unsupervised learning problems consist in finding patterns in the data that are not known a priori. The problems within this class can thus be formulated as the search for a function $f(x_1, \ldots, x_n)$ whose desired function values y_i are not known a priori. The most prominent example of this class of learning problems is clustering. The clustering problem consists in partitioning the available data into groups containing similar instances. The objective is thus to find clusters within the data such that the similarity between instances in the same cluster is high and the similarity between objects of different clusters is low.

3.2 Kernel methods

Kernel methods are one of the most prominent paradigms in modern machine learning research. The core idea of this class of methods is the decoupling of the employed learning algorithms from the representations of the data instances under investigation. Using this paradigm it is possible to use the same learning algorithm for various kind of data, such as vectors, text or graphs. On the other hand, a set of learning algorithms may be applied to one kind of data representation.

The methods for learning from RDF data which will be proposed in Chapters 5 and 6 contribute to the area of kernel methods by proposing data representations which allow to apply kernel methods to RDF data.

The core of kernel methods, the so-called *"Kernel Trick"* is depicted in Fig. 3.1. The data is mapped into some feature space in which the learning problem can be solved. This model which is learnt in the feature space can then be used as a model in the data space. The interesting thing about the feature mapping is that in many algorithms it is not necessary to access the elements of the feature space, but it is sufficient to access the dot product between elements of the feature space. This means that the representation of the data within the feature space need not be calculated explicitely, but can be used implicitely within the dot product. This also enables the handling of feature spaces with an infinite number of dimensions. A more formal definition of kernel functions will be given in Section 3.2.1.

It can be shown that the optimal solution in kernel machines always admits a representation of the form: $f(x) = g(\langle x, x_i \rangle)$, i.e. the solution can be obtained through calculating the dot product with the training instances x_i. It is thus

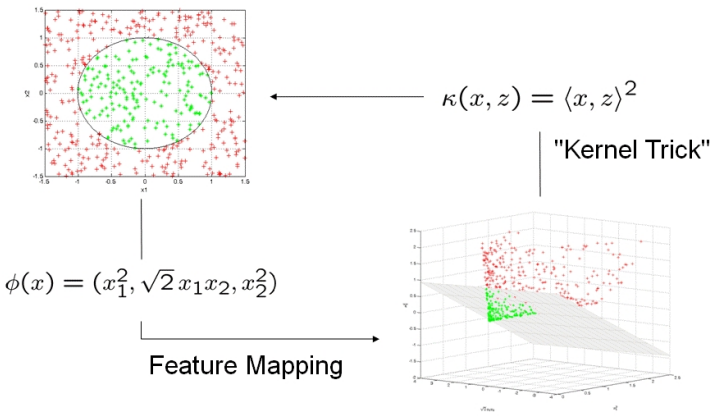

Figure 3.1: Illustration of kernel methods

sufficient that the kernel machine is able to access the evaluations of the inner product $\langle x, x_i \rangle$ of two vectors x, x_i.

As a consequence, it is possible to replace the inner products $\langle \cdot, \cdot \rangle$ in the unkernelized algorithms by any valid *kernel function* which yields the same result as the inner product but can be computed without the explicit representation of the training instances as vectors in the feature space. Thus, kernel machines implicitly mimic the geometry of the feature space by means of the *kernel function*, a similarity function which maintains a geometric interpretation as the inner product of two vectors in some – potentially unknown – feature space.

While SVMs (Vapnik et al., 1997) for classification and regression can safely be regarded as the best known kernel machine, many other well-known supervised and unsupervised machine learning algorithms (e.g. Kernel-kMeans and Kernel-PCA (Schölkopf et al., 1996) for clustering and dimensionality reduction) can be "kernelized" as well. Kernel-based machine learning algorithms abandon the explicit representation of data items in the vector space in which the sought-after patterns are to be detected.

In the following a formal definition of kernel functions will be given and the most important properties will be presented.

3.2.1 Kernel functions

Kernel functions are used to represent data in a form that makes them suitable for their use in kernel machines.

Definition 13 (Kernel Function) *Any function $\kappa : X \times X \to \mathbb{R}$ on some objects x, x' from some input domain $\mathcal{X}$ that satisfies*

$$\kappa(x, x') = \langle \phi(x), \phi(x') \rangle,$$

is a valid kernel, whereby ϕ is a mapping function (feature representation) from $\mathcal{X}$ to a feature space $\mathcal{F}$:

$$\phi : x \to \phi(x) \in \mathcal{F}$$

Intuitively, kernel functions can probably be described best as functions that encode a particular notion of similarity while implicitely maintaining a geometric interpretation. Technically, the set of valid kernel functions exactly corresponds to the set of so-called *positive semi-definite functions* (Shawe-Taylor and Christianini, 2004).

Definition 14 (Positive Semi-definite Function) *Given a set $\mathcal{X}$ and a function $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, then the function κ is called* positive semi-definite *if it is symmetric and if for all $n \in \mathbb{N}$ and $x_1, \ldots, x_n \in \mathcal{X}$, the matrix $K := (\kappa(x_i, x_j)_{i,j}$ is positive semi-definite.*

A symmetric matrix M is called positive semi-definite if and only if for any vector $x \neq 0$, we have $x^\top Mx \geq 0$.

The equivalence of positive semi-definite functions and dot products in vector spaces is given by Mercer's theorem.

Theorem 15 (Mercer's Theorem) *A kernel function κ can be expressed as*

$$\kappa(u, v) = \langle u, v \rangle$$

if and only if, for any function $g(x)$ sucht that $\int g(x)^2 dx$ is finite, then

$$\int \kappa(x, y)g(x)g(y)dxdy \geq 0$$

To show that a function is a kernel function, it is thus either necessary to construct a feature space in which the defined function is the dot product or to show that the defined function is positive semi-definite. A third possibility is to show that the newly defined kernel function can be obtained from functions that are known to be valid kernels using operations under which the space of kernel functions is closed.

Proposition 16 (Closure Properties) *Given two kernel functions κ_1 and κ_2 defined over $X \times X$, $\lambda \in \mathbb{R}^+$, the following functions are also valid kernel functions:*

$$\kappa(x, y) = \kappa_1(x, y) + \kappa_2(x, y) \tag{3.1}$$

$$\kappa(x, y) = \lambda\kappa_1(x, y) \tag{3.2}$$

$$\kappa(x, y) = \kappa_1(x, y)\kappa_2(x, y) \tag{3.3}$$

$$\kappa(x, y) = \kappa_1(x, y)/\sqrt{\kappa_1(x, x)\kappa_1(y, y)} \tag{3.4}$$

The set of kernel functions is thus closed under sum and product, as well as multiplication with a scalar. The last equation presents a so-called kernel normalization *which enables to normalize kernel values to the interval* $[0, 1]$.

Proofs for the presented kernel closure properties and kernel modifiers are for example given by Shawe-Taylor and Christianini (2004).

In general, the use of kernel functions is advantageous in those cases, where the kernel function has better storage or computation requirements than the corresponding explicit feature representation. It is even possible to construct kernel functions corresponding to a feature space with an infinite number of dimensions.

Two well-known kernel functions for vector data are the polynomial kernel and the Gaussian kernel:

Definition 17 (Polynomial Kernel) *The* Polynomial Kernel *of degree p is defined as:*

$$\kappa_{polynomial}(x, y) = (\kappa(x, y) + c)^p, \quad c, p \in \mathbb{R}^+; \tag{3.5}$$

Definition 18 (Gaussian Kernel) *The* Gaussian Kernel *is given by:*

$$\kappa_{gaussian}(x, y) = \exp\left(-\frac{\kappa(x, x) - 2\kappa(x, y) + \kappa(y, y)}{2\sigma^2}\right), \quad \sigma \in \mathbb{R}^+. \tag{3.6}$$

Note that the Gaussian Kernel is an example of a kernel whose (implicit) dimensionality is infinite.

A third frequently used kernel function, which is also the most trivial one, the standard dot product in the current feature space, is known as the *Linear kernel*. The feature mapping implicitely mimiced by the Linear Kernel is $\phi(x) = x$.

Kernel functions may also be defined on other data which is a priori not representable in the form of vectors. The kernel function then maps this data into a vector space and calculates the dot product there. Examples of other data where the representation as vector data is not directly available are graph structures or texts.

3.2.2 Support Vector Machines

The focus of this thesis lies on the application of kernel methods for classification, namely in Support Vector Machines (SVMs). We will therefore not detail on other kernel machines here (the interested reader is referred to (Shawe-Taylor and Christianini, 2004) for an overview of other kernel machines), but will focus on SVMs here. The SVM is an extension of the basic linear classifier; we will therefore start with presenting this classification model before presenting hard-margin and soft-margin SVMs.

Algorithm 1: Perceptron training

Input: Data $S = ((x_1, y_1), \ldots, (x_m, y_m)$ with m instances; Learning rate α

Result: Weight vector w; bias term b

```
w ← 0
b ← 0
repeat
    err ← 0
    for i = 1, ... m do
        compute f(x_i) = sign(< w, x_i > +b)
        if f(x_i) ≠ y_i) then
            w ← w + αy_i x_i
            b ← b + αy_i
            err ← err + 1
        end
    end
until err = 0;
```

Linear classification

The linear classifier is one of the most basic classification models. A classification model in the class of linear classifiers is represented by a hyperplane in the input space, where each instance is classified according to its position with respect to the hyperplane. The problem to solve thus is, given a data matrix X with features $a_1, \ldots, a_m$ and classes $c1$ and $c2$, find a hyperplane described by its normal vector w and the bias term b such that

$$f(x) =< w, x > +b \begin{cases} \geq 0 & \text{if } x \in c_1 \\ < 0 & \text{if } x \in c_2 \end{cases}$$

This means that a hyperplane is searched which separates the data: objects of class $c1$ are on one side of the *separating hyperplane*, objects of class $c2$ on the other.

First applications of linear classification have been studied by Fisher (1936). The first algorithm for learning linear classifiers, the *perceptron learning algorithm* was however only developed in the 1950s (Rosenblatt, 1958). The perceptron training algorithm searches a separating hyperplane given some input data (see Algorithm 1). The algorithm converges to a valid solution if the data is linearly separable.

Definition 19 (Linear separability) *Two sets of points S_1 and S_2 in an n-dimensional vector space are called* linear separable *if there exists a hyperplane*

in this space, such that the points of S_1 are on one side of the hyperplane, the points of S_2 on the other.

Note that if there exists a separating hyperplane, it will be found by the perceptron algorithm. However, the algorithm has no preference with respect to different separating hyperplanes. Support Vector Machines are an alternative way of learning linear classifiers. They learn separating hyperplanes with a maximum distance to the training data. Using the soft-margin version of Support Vector Machines and kernel functions, it is also possible to learn classifiers for data which are not linearly separable.

Support Vector Machine

Support Vector Machines (SVMs) present a set of classification models that has received considerable interest over the last years. This success is due to the remarkable results they achieved on various classification problems, especially on high-dimensional data. They are based on the idea of linear classifiers. Instead of finding any separating hyperplane, SVM finds the separating hyperplane with the maximum *margin*, i.e. with the maximum minimum distance between a training example and the separating hyperplane. The theory of SVMs was first developed by Boser et al. (1992) and refined by Vapnik (1995). A more comprehensible introduction to the subject is offered in Cristianini and Shawe-Taylor (2000).

Definition 20 (Margin) *The* functional margin γ *of a hyperplane* (w, b) *with respect to a data point* (x_i, y_i) *is defined as the quantity*

$$\gamma_i = y_i < w, x_i > +b$$

The geometric margin *is obtained by rescaling w and b. It then represents the Euclidean distance of x_i from the hyperplane:*

$$\gamma_i = y_i < \frac{1}{||w||} w, x_i > + \frac{1}{||w||} b$$

The rationale behind choosing the hyperplane with maximal margin is that classifiers with large margin tend to generalize better than classifiers with a small margin.

Definition 21 (Hard-margin SVM) *Hard-margin SVMs are linear classifiers based on the maximum margin hyperplane. In the case of linear separable data, this hyperplane can be found as solution of the constrained optimization problem:*

$$\min_w \frac{||w||^2}{2}$$
$$\textit{subject to} \quad y_i(< w, x_i > +b) \geq 1, \ i = 1, \ldots, n$$

As the objective function is quadratic and the constraints are linear in the parameters w and b, the optimization problem is known to be convex and can thus be solved using *Lagrangian multipliers*. After reformulation the dual optimization problem is obtained:

Definition 22 (Hard-margin SVM - Dual Optimization Problem)

$$\max_{\alpha} \sum_{i=1}^{l} \alpha_i - 0.5 \sum_{i,j=1}^{l} \alpha_i \alpha_j y_i y_j < x_i, x_j >$$
$$\textit{subject to} \quad \sum_{i=1}^{l} y_i \alpha_i = 0, 0 \leq \alpha_i \forall i$$

It turns out that in the solution most Lagrange multipliers α_i are equal to zero. In fact, if $\alpha_i \neq 0$, the training instance x_i's distance to the separating hyperplane equals the geometric margin. The training instances for which this is the case are called *support vectors*. This situation is illustrated in Figure 3.2.

Note that no solution can be found for the Hard-margin SVM in the case of data which is not linearly separable. The support vector machine as defined above can be extended to be able to deal with data which is not linearly separable. The idea is to relax the margin criterion and to allow data to be wrongly classified. These wrong classifications are associated with a cost which is proportional to its distance from the separating hyperplane. This approach is called *Soft-margin SVM*.

Definition 23 (Soft-margin SVM) *Soft-margin SVMs are an extension of hard-margin SVMs which are able to deal with data which is not linearly separable. It searches the hyperplane minimizing a cost function which penalizes wrong classifications. Given some training data $X = (x_1, \ldots, x_n)$, the optimization problem for the soft-margin SVM is defined as:*

$$\min_{w,b,\xi} 0.5||w||^2 + C \sum_{i=1}^{l} \xi_i$$
$$\textit{subject to} \quad y_i(\langle w, x_i \rangle + b) \geq 1 - \xi_i$$
$$\xi_i \geq 0$$

In this problem, the ξ_i are so-called slack variables *which are proportional to the distance of a misclassified example from its corresponding margin.*

The choice of the cost factor C is crucial in soft-margin SVM as it determines the trade-off between a large margin and the number of misclassified instances. Choosing a large C makes the misclassification of instances expensive, while a small C leads to more tolerance towards misclassified instances.

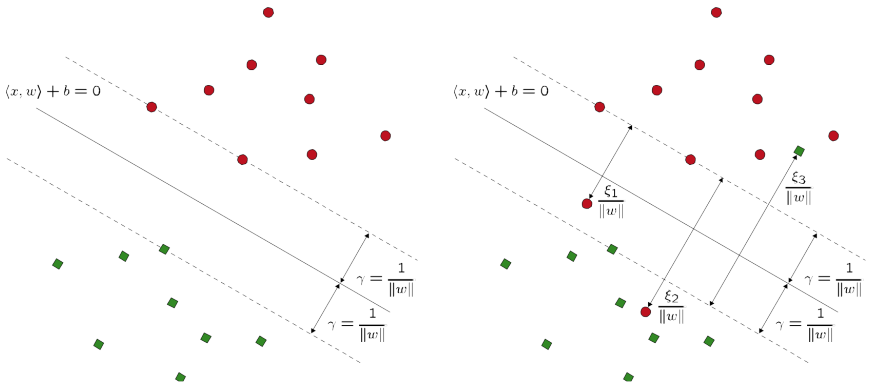

Figure 3.2: Separating hyperplane in the case of hard margin SVMs (left side) and in the case of soft-margin SVMs (right side). In the case of hard-margin SVM, the hyperplane which maximizes the margin is denoted by a solid line, the lines which fix the margin are shown as dotted lines, the points on the margin are the support vectors. In the case of the soft-margin SVM, some points lie on the wrong side of the separating hyperplane.

Definition 24 (Soft-margin SVM - Dual Optimization Problem) *The dual version of the optimization problem is obtained as:*

$$\max_{\alpha} \sum_{i=1}^{l} \alpha_i - 0.5 \sum_{i,j=1}^{l} \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle$$

$$\textit{subject to} \quad \sum_{i=1}^{l} y_i \alpha_i = 0, 0 \leq \alpha_i \leq C \forall i$$
$$\xi_i \geq 0$$

In practice, the optimal choice of the parameter C is very important as it heavily influences the obtained results. However, the interval from which the values for C may be chosen is very large and no approach for identifying good C-values is available. Schölkopf et al. (2000) have proposed an alternative formulation of the SVM learning problem which replaces the parameter C with a new parameter ν which has a clearer interpretation:

Definition 25 (ν-SVM) *The optimization problem of the ν-SVM is defined*

as:

$$\max_{\alpha} -0.5 \sum_{i,j=1}^{l} \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle$$

$$\textit{subject to} \quad 0 \leq \alpha_i \leq 1/l$$

$$\sum_{i=1}^{l} \alpha_i y_i = 0 \qquad \sum_{i=1}^{l} \alpha_i \geq \nu$$

The newly introduced parameter ν takes values between 0 and 1 and has a clearer interpretation as C: Schölkopf et al. (2000) have shown that

- ν is an upper bound on the fraction of margin errors, i.e. on the fraction of instances which are wrongly classified or whose distance to the classification border is smaller than the margin of the classifier.
- ν is a lower bound on the fraction of support vectors.

This means that ν allows for the direct control of the model complexity.

3.3 Learning from Graphs

So far we have focused on methods for mining vectorial data. However, Semantic Web data is not intuitively representable as vectors. Instead, at least in the case of RDF, an intuitive representation in the form of graphs is available. The methods presented in Chapter 5 and 6 use a graph representation of RDF data.t

Graphs are a means to represent all kinds of elements and relations between them. Typical examples of data that is represented in the form of graphs are social networks and chemical compounds. In this section we will introduce the basics of graph theory and we will give an overview of graph classification algorithms. For a more complete overview of graph mining algorithms also for other problems, the interested reader is refered to (Martino and Sperduti, 2010).

The problem which has probably been studied most extensively in the context of graphs, is mining of (approximate) *frequent graph patterns*. The task is to find a set of subgraphs within a single graph or a set of graphs, which occur frequently. However, in the context of this work, we will focus on another problem: classification with graphs as input data.

In the following sections, we will first introduce some basic concepts from graph theory and in a second step, we will present approaches for graph classification.

3.3.1 Basic Graph Theory

The instances in graph mining are represented by graphs. In the following, we will introduce the definitions of graphs and graph structures which are relevant for the remainder of this thesis. A more in-depth introduction is presented by Goos (2000).

Definition 26 (Graph) *A* directed graph $G = (V, E)$ *is definded by a set of nodes (also called* vertices*)* V *and a relation* $E \subseteq V \times V$*. The elements of* E *are called* edges*.*

If E *is symmetric,* G *is called* undirected*.*

Graph mining is based on finding and counting specific patterns within graphs. The most important structural patterns are walks, cycles and paths.

Definition 27 (Walk, Cycle, Path) *A sequence* $(e_0, e_1, \ldots, e_n)$ *of edges* $e_i = (v_i, v_{i+1}) \in E, i = 0, \ldots, n-1$ *is called* walk *of length* $n \geq 0$*.*

A walk of length $n \geq 1$ *is called a* cycle*, if* $v_0 = v_n$ *and if* $e_i \neq e_j, 1 \leq i, j \leq n$*. A cycle is called Eulerian, if it contains each element of* E *exactly once. A cycle is called Hamiltonian, if it contains each element of* V *exactly once. A graph is called* acyclic*, if it does not contain any cycles.*

A walk that does not contain any cycles, is called a path*.*

Trees are specific kinds of graphs which are defined based on their structural elements. They are a class of graphs that can often be handled much more efficiently than general graphs. Therefore, it is often interesting to reduce general graph structures in such a way that trees are obtained.

Definition 28 (Tree) *An* undirected tree *is an acyclic graph, in which there is exactly one walk between each pair of nodes. In undirected trees the number of nodes is obtained as* $|V| = |E| + 1$*.*

A directed tree *is an acyclic graph in which the number of incoming edges of each node is smaller or equal to* 1 *for all nodes and equal to* 0 *for exactly one of the nodes, the so-called* root node*.*

In ML, similarity measures and equality of elements are very important. However, there is no direct notion of equality on graphs, as nodes are in general not identifiable. It is only possible to identify nodes which have the same labels and the same neighborhood. Two graphs are equivalent, i.e. they have the same structure, if there is an isomorphism between the two graphs:

Definition 29 (Graph isomorphism) *A* graph morphism *is a function* $f : G_1 \to G_2$ *which maps from one graph* $G_1 = (V_1, E_1)$ *to another graph* $G_2 = (V_2, E_2)$*, i.e.* $f(v) \in V_2 \forall v \in V_1$ *and* $(f(v_1), f(v_2)) \in E_2 \Leftrightarrow (v_1, v_2) \in E_1$*. Two graphs* G_1*,*G_2 *are called isomorphic, if there exists a bijection between the two graphs, i.e. a morphism of* G_1 *into* G_2 *and vice versa.*

The problem of deciding whether two graphs are isomorphic is known to be in NP, no polynomial testing procedure is known (although NP-completeness has not been shown either). In the context of mining graphs isomorphism checking is therefore to be avoided.

Research on graph mining can be distinguished in two classes: either the learning problem works on a set of graphs (i.e. each instance is a graph) or the learning problem deals with subgraphs of a single graph. The problems we are concerned with are part of the first class. For an overview of the second class the interested reader is refered to (Chakrabarti and Faloutsos, 2006).

3.3.2 Graph classification

A graph classification problem is a classification problem, where the input data is described by graphs. Given some input data $(x_1, y_1), \ldots, (x_n, y_n)$ where the x_i are graphs and the y_i values of a class variable, the problem consists in finding a function f such that $f(x_i) = y_i$.

In general, graph classification uses the same learning algorithms as proposed in vectorial data mining. Especially kernel methods have received considerable interest when dealing with graphs due to their decoupling of the learning algorithm from the data description. Given a suitable kernel function, kernel machines are able to deal with graph data. The advantage of kernel machines over other learning algorithms in this context is that there is no need for explicitely representing the features of the feature space and therefore potentially very high-dimensional feature spaces may be defined.

The challenge in any graph classification problem consists in finding suitable features to describe properties of the graph and fast algorithms for calculating the instance representation in the feature space resp. the value of the corresponding kernel function. Note that in the case of kernel methods the representation in the feature space is not made explicit. However, the input graph is mapped to a feature space implicitely in this case.

In the following, we will give an overview of the structures that have been used as features in graph classification.

(Connected) Subgraphs

Shervashidze et al. (2009) have proposed the use of small connected subgraphs with 3 to 5 nodes as features. The idea is that a graph is best described by its subgraphs. While checking whether a graph contains a specific subgraph is expensive, the decomposition of graphs into its subgraphs of a specific size can be done more efficiently and isomorphism checking is feasible for small graphs. The kernel then works by determining for any pair of graphlets of size $k \in \{3, 4, 5\}$ whether they are isomorphic.

Trees

Shervashidze and Borgwardt (2009) have proposed to use neighborhood trees, i.e. the neighborhoods up to k hops from each node as features. They have proposed an efficient algorithm for calculating the kernel function based on this feature mapping: starting from trees with one node, an iterative algorithm expands the size of the trees up to the maximum depth by encoding the neighborhood of each node into the node in each step. Each distinct node label then represents one feature.

Horváth et al. (2004) use tree patterns as one part of their cyclic pattern kernel which will be explained below.

Costa and Grave (2010) have proposed to use pairs of identical trees with depth r and distance d between the trees in the graph as features. The kernel function based on the feature mapping counts all such pairs of trees up to some maximum depth r and some maximum distance d.

In tree mining, tree structures are also often used as features. The idea is that trees are best described by their subtrees. This kind of approach has for example been proposed by Vishwanathan and Smola (2003). Moschitti (2006) has studied subtree kernels for dependency and syntactic trees. The specific problem of this kind of trees is that the order of the leaves is fixed. The author has proposed efficient methods for calculating kernels based on subtrees as well as partial subtrees as features.

Walks, Paths and Cycles

The first graph kernels were based on graphs and walks as features: Gärtner et al. (2003) have proposed to use the set of walks up to infinite length in the graph as feature space. They show that the common walks can be obtained as the limit of a power series of the product graph's adjacency matrix. However, calculating this limit involves inverting the adjacency matrix which makes the calculation expensive for large graphs. It has been noted that these random walk kernels suffer from *tottering*, meaning that very small common substructures can lead to high similarity values. This is especially the case for cycles of nodes that are visited again and again.

Borgwardt and Kriegel (2005) have proposed a feature space which consists of pairs of nodes and the length of the shortest paths between them in the graph. The kernel they propose works on graphs which have the same nodes as the input graph, but which contain an edge between two nodes if there exists a walk from the source to the target node of the edge. The edges are labeled with the length of the shortest path from source to target. The kernel then compares all pairs of edges using any valid kernel for walks of length 1.

The use of cycles as features has been proposed by Horváth et al. (2004). The feature space they propose consists of all cycles that are found in the set of

graphs under consideration. However, the authors find that information on cycles is not sufficient as only type of structure inducing similarity between graphs. Therefore, tree patterns are used as additional features: all edges belonging to a cycle in the graph are deleted from the graph. As result a forest (a graph consisting of one or several trees) containing all bridges in the graph is obtained. Any tree pattern found in this forest is used as additional feature in the kernel.

Frequent subgraphs

Another approach for defining interesting features consists in searching for frequent subgraphs within the set of graphs, which are then used as features in the learning task. As stated above, finding frequent subgraphs is a problem which has been studied extensively in the literature. The methods developed in this line of research can then be used for finding features for classification. This approach has for example been used by Kudo et al. (2004): They use gSpan (Yan and Han, 2002) for finding frequent subgraphs and use these as basis for decision stumps which are then combined in a boosting approach to learn a classifier.

Deshpande et al. (2005) have used a similar approach in the context of chemical compounds: the features they propose are based on searching frequent topological and geometric patterns in a set of compounds which are then used as features for classification.

Domain-specific features

While the previously presented approaches use specific structures in the graph as features, there are also approaches to use specific domain-dependent features for graph mining. For example, Bloehdorn and Sure (2007) have defined kernels that work specifically on semantic data. The instances in their kernel functions are instances (i.e. specific nodes in the underlying graph). However, their kernels are not directly defined on the graph structures underlying the data but on the semantic level. For example, they have defined kernels based on the identity of two instances, on the classes the instances share, on the data properties the instances share and so on. Besides the identity kernel these kernels could also be defined on the graph obtained after materializing the ontology. For example, the common class kernel uses links labeled with the type relation as features.

In a different context, Fröhlich et al. (2005) propose to define kernels for comparing atoms within molecules. They propose to use the role of the atom as well as structural features, such as whether an atom is part of a cycle as features. Their contribution then consists in defining a kernel which allows for the integration of the information available for each atom into a kernel for whole molecules.

Part III

Ontology Change Support

Chapter 4

Ontology Change

Ontologies are hard to develop as they do not only require a lot of domain knowledge but also require the knowledge of how the available domain knowledge can best be described in a specific formalism. There are many methods that aim at helping the domain expert formalize all the knowledge that is necessary to get a complete description of the domain, that help detect problems (e.g. inconsistencies) in the ontology and that help the ontology engineer to adapt the ontology to changes in the requirements and the modeled domain. The goal of these methods is to improve the quality of the resulting ontology and to facilitate the task of the ontology engineer. All described activities are subsumed by the term *Ontology Change*.

Definition 30 (Ontology Change) *Ontology Change refers to the problem of deciding the modifications to perform upon an ontology in response to a certain need for change.*

According to this definition any change to an ontology is covered by the term Ontology Change. The change in the ontology is triggered by the detection of a need for changing the ontology (this need may stem from various sources, see Section 4.1.1 for more details) and is concerned with acquiring the additional knowledge which is needed for implementing the change and integrate the new knowledge in the ontology (see Section 4.1.2 for more details). The concrete changes are then performed on the formal specification by e.g. adding, removing or changing the definition of concepts, relations or properties or by adding, removing or changing the description of a concrete instance in the ontology. The definition of Ontology Change thus goes beyond the mere application of a set of changes to the ontology at hand, it also covers the process of defining the set of changes to perform, i.e. to decide which information in the ontology should be changed in order to adapt the ontology according to the need for change which has been detected.

Dimension	Values
Reason of Change	Ontology Refinement Ontology Evolution Ontology Update
Change phase	Problem Detection Knowledge Acquisition Knowledge Integration

Table 4.1: Overview of the classification schema for Ontology Change methods

Flouris et al. (2008) give a broader definition of the term: they consider that the implementation of the changes made to the ontology in dependent applications and services is also part of the change process. According to this definition, Ontology Change is not only concerned with the change of the ontology itself but also with the adaptation of systems in which an ontology is used to the new version of the ontology. This thesis focuses however on supporting changes to the ontology itself and does not take the change of dependent applications into account.

The remainder of this chapter is structured as follows: Section 4.1 presents a classification of problems in the context of changing an ontology. Section 4.2 gives an overview of existing approaches to support ontology change tasks and Section 4.3 discusses the existing approaches, identifies shortcomings and motivates our contributions from them.

4.1 Dimensions of Ontology Change

In the following, we will define a classification schema for methods which support ontology changes: the dimensions in this schema are the reason for changing the ontology and the phase in the change process which is supported. An overview of the proposed classification schema is given in Table 4.1.

4.1.1 Reason of Change

The first dimension of an ontology change is the motivation for changing the ontology. Ontologies may be changed because they are still in the process of being created, because their requirements have changed, because the domain they model has changed or because some problem within the ontology itself have occurred.

We distinguish three reasons for changing an ontology:

- *Ontology Refinement* deals with obtaining additional information which should be part of the ontology and to integrate it. The goal is to obtain a

more complete and more fine-grained ontology (Sure et al., 2002).

- *Ontology Evolution* addresses changes in the ontology due to changing requirements. With the use of an ontology, the focus of its application may change or additional scenarios may be addressed. Ontology Evolution deals with adapting the ontology to these changes in requirements (Stojanovic, 2004).

- *Ontology Update* adapts the ontology to changes in the domain itself. This means that neither the requirements have changed nor the application, but the domain itself (Katsuno and Mendelzon, 1992; Lösch et al., 2009).

The changes issued by each of these change scenarios may be similar, however the reason for which the ontology was changed is different. Consider the introduction of a new class `EU-country` into an ontology. If the concept is added because it was missing in the ontology, although it was needed in the application, then the change is a case of *Ontology Refinement*. The concept may also be added because in the original version of the ontology only countries in general were modeled and now the ontology is also used for an application where it is important to distinguish between the countries belonging to the European Union and those that don't. This means that the requirements for the ontology have changed and the change is a case of *Ontology Evolution*. The third reason for which the concept `EU-country` may be added to the ontology is because the European Union has not existed before. In this case the change occurs becaus the underlying domain has changed and we are dealing with a case of *Ontology Update*.

Note that the distinction between these scenarios is frequently not made in the literature. While e.g. the terms Ontology Evolution or Ontology Update are often used in the literature, the clear distinction between the three scenarios described above is usually not made. E.g., while Flouris et al. (2008) come up with a set of tasks that have to be dealt with in the context of Ontology Change, they subsume any activity which is related to changing an ontology itself under the term Ontology Evolution. However, each of the scenarios described above yields different requirements with respect to the approaches developed for supporting them. We therefore think that they should be distinguished conceptually.

4.1.2 Ontology Change Process

The second dimension according to which methods for supporting ontology change may be classified is the task in the change process they address.

There are three main phases of Ontology Change which form a cycle for continuous Ontology Change (see Figure 4.1). These three tasks are Problem Detection, Knowledge Acquisition and Knowledge Integration.

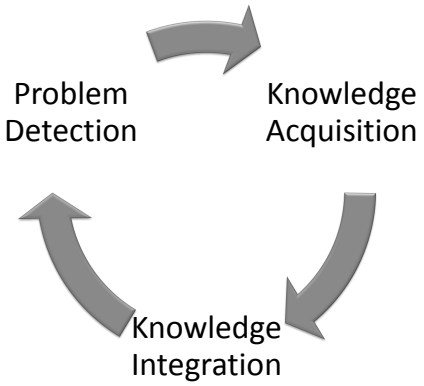

Figure 4.1: The Ontology Change Cycle

- In the *Monitoring and Problem Detection* phase, the ontology and the domain are observed. When the need for updating, completing or correcting information in the ontology is detected, the change process may be triggered.

- In the *Knowledge Acquisition* new knowledge is acquired. The source of this new knowledge may be the ontology itself, it may be extracted from external data, or it may be obtained from the domain engineer.

- Finally, the obtained knowledge is integrated in the ontology in the *Knowledge Integration* phase. The new knowledge may be inaccurate or it may contradict knowledge which is currently modelled in the ontology. Thus, decisions with respect to which parts of the ontology should effectively be changed are necessary. The result of this phase is a consistent and coherent ontology incorporating a solution to the problem which initiated the change. Consistent here refers to logical consistence, coherence refers to the fact, that no unintended knowledge should be part of the ontology.

These phases are independent of the motivation for changing the ontology. However, different actions may be required in each of the phases and different data may be analysed. For example, the need for refining the ontology can be detected by analysing the ontology and related requirements while the need for an update is detected from the combination of the ontology and the underlying domain. The detection of a problem triggers the change cycle in which new knowledge is acquired and added to the ontology. Again, in these two phases the methods employed may depend on the reason of the change and the additional data which is available.

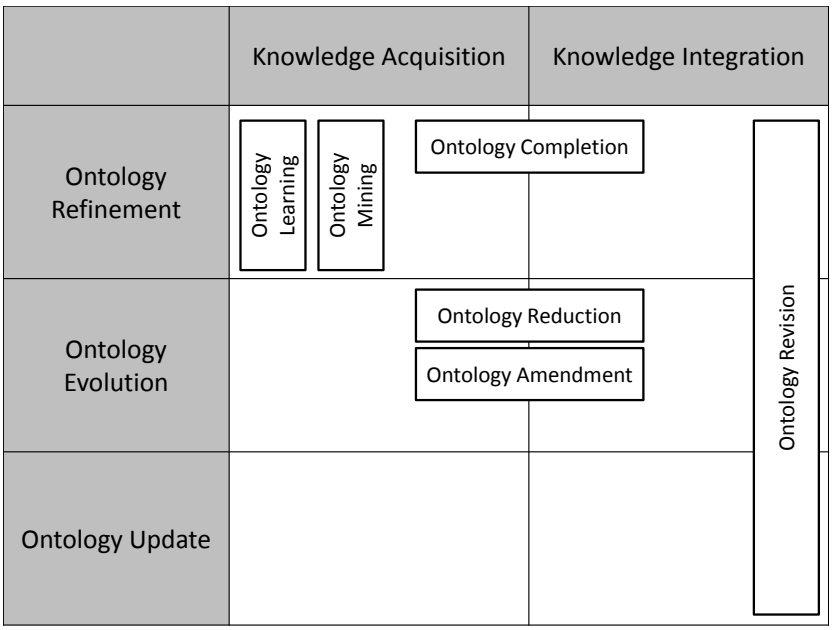

Figure 4.2: Classification of some research fields with respect to their coverage of Ontology Change

4.2 Existing change support methods

There has been a lot of work on supporting changes in an ontology and whole fields of research are addressing specific issues arising from the different kinds of changes that occur in an ontology. For example, the fields of Ontology Learning and Ontology Mining address the problem of Knowledge Acquisition in Ontology Refinement. An overview of relevant directions of research and their position with respect to the ontology change process and reason of change is given in Figure 4.2.

These fields of research, which will be presented in more detail in the following, are:

- *Ontology Completion*, which deals with the acquisition and integration of additional knowledge into the ontology with the goal of obtaining a complete axiomatization of the domain,
- *Ontology Mining* which is concerned with analysing the current ontology and inducing additional knowledge which may be used for extending the ontology,
- *Ontology Learning* which uses the results of analysing external - textual - documents in order to obtain a more concise and stronger axiomatised description of concepts in the ontology.

- *Ontology Revision* which is concerned with the integration of new knowledge into an ontology while keeping some of the ontologies' properties, e.g. its consistency,
- *Ontology Amendment* which is concerned with automatically acquiring new axioms in order to adapt the ontology to new use cases, and
- *Ontology Reduction* whose aim is to reduce the ontology's size during the evolution process, e.g. by eliminating knowledge which is infrequently used.

In the following we will discuss each of these directions of research in more detail and present the main ideas that have been proposed to address the issues raised in the specific fields.

4.2.1 Ontology Completion

The field of Ontology Completion is focussing on methods for determining missing information in the ontology and to help domain experts create a complete representation of the domain. One motivation is that domain experts often struggle to add all the information that is necessary to obtain unambiguous and complete domain descriptions. To help the developer in this process, methods for detecting missing information and for amending the missing parts in the ontology have been developed.

The most important line of research is the adaptation of Formal Concept Analysis (FCA) for ontology languages. There, an interactive process for exploring the domain of interest is proposed where any statement which can be deduced from the ontology is validated resp. rejected automatically. The user is asked for the correct answer in cases where no decision can be made automatically.

Formal Concept Analysis (FCA) (Ganter and Wille, 1999) deals with finding implications in so-called formal concepts, i.e. relations describing the memberships of given objects in given concepts. Given a partial description of the domain of interest, FCA helps in acquiring a complete description of this domain, i.e. knowledge about any implication holding between concepts.

Traditionally, FCA deals with closed worlds, i.e. with the case where a fact can either hold or not hold, there is no unknown state. However, the method has been adapted for the use with Description Logic (DL) ontologies with an open-world semantics, i.e. where the absence of a statement does not mean that it does not hold (Rudolph, 2006; Baader et al., 2007). This is achieved by trying to decide for any unknown implication whether it holds or can be refuted using a reasoner. If no answer can be obtained from the reasoner, the user is asked to add some information which allows to accept or refute the implication.

The basic approach was refined in Sertkaya (2009). Usability issues in the approach were addresed and the possibilities of taking back some decisions and postponing specific questions were added. *Relational exploration* extends the ideas from the learning of hierarchies for simple concepts to the case of General Concept Inclusions, i.e. to the case where not only atomic concepts have to be dealt with but also complex concept descriptions like intersections or unions of concepts (Rudolph, 2006).

This class of approaches thus relies on deductive approaches and the interaction with a user. However, additional information may be obtained using inductive methods which allow to infer additional knowledge and thus take more burden from the user, as they give indications with respect to the truth values of certain statements or because they propose examples which may be used for contradicting certain statements. This kind of approach has been proposed e.g. by Völker and Rudolph (2008). This kind of integration with the analysis of external data will be discussed in Section 4.2.3.

4.2.2 Ontology Mining

Ontology Mining deals with the analysis of ontology data with inductive means in order to obtain new information which is not yet present in the ontology. Two classes of approaches may be distinguished in the field: one class of methods is based on Inductive Logic Programming (ILP) and tries to identify logical formula covering specific sets of instances, and the second class uses statistical learning methods to derive models for problems like link prediction.

A principled method for integrating data mining models with RDF data has been proposed by Kiefer et al. (2008). They have proposed a framework which allows for the definition, induction and scoring of data mining models within the SPARQL query language. New language constructs are introduced into the SPARQL language for this purpose. They do not propose specific mining algorithms but only a general approach how the models constructed using any learning algorithm may be integrated into a SPARQL endpoint.

Learning based on Inductive Logic Programming

The ILP approach is based on the analysis of the extension of some concept to be described (i.e. the list of instances) and in finding intensional descriptions, i.e. logical formulae, describing the same set of instances in the given knowledge base (Muggleton and de Raedt, 1994). This means that the learned concept description should cover all the instances in the instance set, but no instance which is not part of the predefined set. The problem is also known as *Concept learning*. Note that in order to derive descriptions which hold in general and not just in the given state of the knowledge base, examples for all relevant

phenomena have to be present in the knowledge base.

The general approach for learning concept descriptions consists in generating candidate descriptions which can then be evaluated using a standard reasoner. In order to obtain results in an efficient manner, the possible concept descriptions have to be ordered in a way such that the generated candidate descriptions are promising ones and such that the search space can be pruned efficiently.

To generate candidates, Lehmann (2007) propose the use of genetic programming for candidate generation. Candidates are evaluated using a fitness measure which expresses how well a concept description covers the set of instances of interest. However, the standard operators used in genetic programming are problematic in the case of learning concept descriptions as the crossover operator tends to be too destructive, i.e. the newly generated solution is too far from the original solution and thus the new solution is usually worse than the original one. Therefore, new refinement operators are introduced which allow the generalisation as well the specialisation of a concept description (Lehmann, 2007). Lehmann and Hitzler (2007) have defined a refinement operator for the Description Logic $\mathcal{ALC}$ which is optimal in the sense that it has as many of the properties desired in a refinement operator as possible.

The efficiency of the learning heavily depends on the choice of a suitable DL fragment which is expressive enough to obtain the searched concept description while being efficient with respect to the reasoning necessary to check the quality of a candidate concept description Hellmann et al. (2009). To process large knowledge bases, it may also be desirable to only use fragments of the original knowledge in the learning Hellmann et al. (2009).

Lehmann (2009) provides a framework for and an implementation of ILP-based learning methods which implements many of the methods for ontology mining described so far.

An approach to use ontologies in Relational Learning using ILP has been proposed by Lisi and Esposito (2008). Their approach relies on the integration of ontologies with disjunctive DATALOG and define an algorithm which learns horn rules in this formalism based on the available extensional knowledge.

Cimiano et al. (2010) define an approach for finding intensional answers to answer queries to an ontology. The idea is to find a concept description which covers exactly those instances which are returned as answer to the posed query. Starting with formulae describing the single answers, the approach consists in a stepwise generalization of the description, resulting in a concept description which covers all the returned answers. While the approach is originally proposed as a way to better describe query answers, it may also be used for ontology refinement, as the descriptions which were found using the approach, may also be added to the ontology.

The general problem of ILP-based approaches is however that they require

that the model which is sought, e.g. the concept to be described may be expressed in the given ontology. If the existing concepts are too general to describe the concept to be learnt, no solution may be found using these approaches. An alternative to the ILP approaches which learn within the existing ontology are statistical methods which will be presented in the following section.

Adaptation of Machine Learning Methods

In contrast to the ILP-inspired methods presented above, there are also many approaches which are based on statistical analyses of the available ontology data.

These methods adapt classical machine learning methods which usually were developed for working with relational database data, such as the k-nearest neighbors method to work with ontology data.

k-nearest Neighbor Approaches. Fanizzi et al. (2007) have adapted a k-nearest neighbor approach for learning class assignments in ontologies by defining a suitable similarity measure between individuals from the ontology. The similarity measure is based on concepts to which the individuals do or do not belong. The set of atomic concepts and any other concept defined explicitely in the knowledge base is a set of features which enables good performance of the classifiers (d'Amato et al., 2008). The rationale for these similarity measures is that the similarity between individuals is determined by their similarity w.r.t. each concept in a given committee of features (a sort of similarity *context*). Two individuals are maximally similar w.r.t. a given concept if they exhibit the same behavior, i.e. both are instances of the concept or of its negation. Because of the *open-world* semantics, a reasoner may be unable to ascertain the concept-membership of some individuals, hence, since both possibilities are open, an intermediate value is assigned to reflect such uncertainty.

Kernel methods. In a different line of research kernel functions for Semantic Web data have been proposed. Kernel methods are a promising candidate for learning from this kind of data as they separate the data representation from the learning algorithm and once suitable kernel functions for some kind of data have been defined, the whole family of learning algorithms may be applied to this data representation. In principle, kernel methods thus present a general framework for learning in the context of the Semantic Web, as they allow for the direct application of existing methods in a Semantic Web context.

Kernel functions for Semantic Web data rely on the same ideas as kernel functions for data represented in any logical formalism. The basic idea presented by Gärtner et al. (2004) consists in defining kernel functions based on type construction, where types are defined in a declarative way. Given a set of type constructors, they propose to define one kernel per type constructor. The thus-

defined kernels can then be combined using kernel modifiers such as sum and product.

The first kernel functions for semantic web data were restricted to the basic description logic $\mathcal{ALC}$ (Fanizzi and d'Amato, 2006, 2007). These kernel functions compare instances based on the structural similarity of the AND-OR trees corresponding to a normal form of the instances' concept descriptions (Baader et al., 2007). Their applicability is restricted due to the employment of the notion of (approximations of) *most specific concepts* (Baader et al., 2007) in order to lift instances to the concept-level where the kernels actually work. Additionally, the normal form of the concept descriptions is specific to the employed description logic. However, these kernels are not purely structural since they ultimately relies on the semantic similarity of the *primitive* concepts assessed by comparing their extensions (approximated by their retrieval) through a set kernel. Structural kernels for richer DL representations have been proposed by Fanizzi et al. (2008a). Here, the kernels from Fanizzi and d'Amato (2006) and Fanizzi and d'Amato (2007) were extended to cover $\mathcal{ALCN}$.

A definition of kernel functions for individuals in the context of the standard Semantic Web representations is reported by Bloehdorn and Sure (2007). The authors define a set of kernels for individuals based on their similarity with respect to the various kinds of assertions in the ABox (i.e. with respect to common concepts, datatype properties and object properties).

A more flexible way of defining kernel functions is based on simple similarity functions parameterized on the semantics of instances w.r.t. a committee of concepts. Such kernels can be integrated with many efficient algorithms, that can implicitly embed feature selection. These functions transform the initial representation of the instances into the related - so called - active features, thus allowing for learning the classifier directly from structured data (Cumby and Roth, 2003). In this spirit, a different set of kernels, which is directly applicable to individuals, has been proposed by Fanizzi et al. (2008b), adopting the ideas of the similarity measure defined by d'Amato et al. (2008). This approach for defining kernel functions rely on a well-modelled domain in which features may be distinguished according to the concepts they belong to. Information which is not expressed by means of a concept is not taken into account by this approach.

Bicer et al. (2011) have proposed kernel methods for links in RDF graph which are based on the grounding of specific ILP clauses in the RDF data graph. The actual kernel function is learned as a non-linear combination of simple clause kernels. However, their learning approach does not allow for the direct use of the proposed kernel function in any kernel machine. Instead, an adaptation of the learning algorithm for choosing the adequate clause kernels is needed.

Statistical Learning. In the context of the Semantic Web, Statistical Learning approaches are mainly applied to the Link Prediction problem which consists in deciding whether a link of a specific type should exist between two entities of interest.

Matrix-based approaches for solving this problem define a relation matrix which contains the elements of the domain of the relation as rows and the elements of the range as columns. Additional knowledge for the domain instances may be added as additional columns. The entries of the relation matrix are random variables expressing whether there is a link of the desired type between the two entities. Matrix-completion methods such as Singular-Value Decomposition are then applied to the matrix to obtain predictions for the unknown values (Tresp et al., 2009; Huang et al., 2010). The matrix completion approach relies on the instances of the learned relation itself as well as on small parts of the direct neighborhoods of the relation subjects. Additional information which could help the approach, e.g. on the similarity of the objects, is not used in this approach.

Relational Graphical Models provide a different approach for statistical modeling of the domain of interest. A belief network is defined where each possible statement is represented by a random variable and the prediction consists in finding the assignment of truth values to the random variables which yields the highest propability and thus is most likely to be true. Rettinger et al. (2009) use this kind of approach for link prediction in semantic data. Their approach allows for exploiting formal domain knowledge as prior knowledge in the belief network. An interesting aspect of this approach is that while learning one relation it is also possible to determine missing statements for other relations using this approach.

Association Rules. Völker and Niepert (2011) have proposed an approach for inducing a schema for RDF knowledge bases. Based on the concepts and relations present in the ontology, axioms are obtained using a frequent itemset mining approach. The axioms are derived from a transaction database which contains for each instance the information to which concepts it belongs. Association Rules are mined from this transaction database, where each found association rule indicates a probable concept inclusion. The axioms learned using this approach include class and property hierarchies as well as domain and range desctrictions.

4.2.3 Ontology Learning

Ontology Learning from lexical resources has received a lot of interest as textual descriptions of objects or concepts are often available and easier to state than the equivalent logical formula. Additionally, Information Extraction is a

wide field in which many methods now used in the context of ontologies have been developed before ontologies have become a popular means for knowledge management.

The *LExO* approach (Völker et al., 2007a) aims at refining the definition of concepts for which a textual definition is available. The method works by applying a set of manually defined rules to translate the description text into a set of logic formulas. An example for a rule would be that two parts of the description linked by "or" form a union in the resulting formula. This approach relies on clear definitions of the involved concepts and relations in a formal language. This means that the effort needed for modelling an ontology is reduced only by the factor of writing the logical formula representing the concept, the definitions themselves have still to be written by the ontology engineer.

Cimiano et al. (2005) have employed a combination of text analysis and FCA for learning concept hierarchies. The available text corpus is parsed and a formal context is constructed where the noun phrases from the corpus are used as objects and the verbs they cooccur with are used as attributes. Using this formal context a concept lattice can be derived from which the result concept hierarchy is obtained after pruning.

Völker and Rudolph (2008) have proposed the integration of the LExO approach (Völker et al., 2007a) for learning concept descriptions with Relational Exploration (Rudolph, 2006). The goal of this approach is to refine an ontology with respect to a specific concept. It is assumed that a textual description of the concept is available which can be used for obtaining first descriptions for the new concepts. Relational Exploration can then be used in a second step on a predefined set of concepts to clarify their relation to other elements of the ontology. This approach integrates ideas from ontology completion and ontology learning. Still, the formal definition of the learned concept is necessary. However, the Relational Exploration facilitates the seamless integration of the new concept with the existing ontology.

Velardi et al. (2005) have proposed a method for ontology learning which is less guided and may start ontology learning from scratch. The first step of the approach consists in finding relevant terms in a given corpus. Then, definitions of these terms are sought on the web. If such definitions are found they can be used for refining the concept they define. If no suitable definition is found for complex expressions but definitions for their components are found, these partial definitions can be used for the definition of the complex concept after it has been found how the partial concepts are related in the complex concept name. If no suitable definitions are found on the Web even for parts of the concept name, then WordNet (Fellbaum, 2005) is used as resource for selecting the appropriate senses for the concept terms. This approach allows for a fully automated learning approach. However, this may lead to ontologies of low

quality and a lot of manual work may be needed for obtaining an acceptable ontology. Therefore, interactive semi-automatic approaches may be preferred in many cases.

Buitelaar et al. (2004) propose to extract concepts from text through the manual definition of so-called mappings. A mapping here is a rule which defines how a class may be identified in the text. The rules may not only depend on the lexical analysis of the text, but may also base on tokenization, part-of-speech tagging, morphological analysis and lexical semantic tagging. The drawback of this approach is that it requires a precise definition used for the extraction of instances, which also rely on linguistic concepts requiring the domain engineer to be familiar with these concepts for modelling the domain. However, the approach presents a trade-off between the approach by Velardi et al. (2005) which is completely unsupervised and the approach by Völker and Rudolph (2008) which relies on formal textual definitions of the concepts.

Cimiano and Völker (2005) have proposed a framework which extends the above-described approaches with a probabilistic ontology model. The learned axioms are not represented using a specific ontology model like OWL or F-Logic but using a set of language-independent modeling primitives. This generic model can easily be translated into various ontology models. Each of the found axioms is associated with a probability of its holding in the ontology. Thus, each learned statement has a confidence value attached which helps the domain engineer decide which of the axioms should hold in the learned ontology. This approach is interesting as it takes the varying quality of the identified concepts and relations into account and presents the confidence of the system in some learned knowledge to the user.

While most of the presented approaches focus on relatively light-weight axioms, Völker et al. (2007b) have proposed a method for learning more complex axioms, specifically disjointness axioms. They propose to learn a classifier for deciding whether two classes are disjoint. The features are based on information from the ontology, such as common subclasses of the two classes of interest or subsumption relations between the two classes. Additionally to the measures of similarity taken from the ontology itself, a text corpus is used for obtaining additional evidence: e.g., if two classes occur in the same enumeration, their likelihood of being disjoint increases.

4.2.4 Ontology Revision

Ontology Revision deals with the integration of a set of newly acquired axioms into an existing ontology. Here, the goal consists in adding new knowledge resp. retracting knowledge from an ontology in a way that the result is a consistent ontology which incorporates the new information.

Stojanovic (2004) have proposed so-called evolution strategies which are defined for each possible type of change. The evolution strategy defines the additional changes that may have to be applied and the problems that may occur when a change is applied and thus guide the detection and resolution of problems occuring in the change process. While this approach allows for the completely automatic change integration, the considered changes are identified on a purely structural level which does not take information from the domain into account.

In the same spirit of completely automatic change integration a set of approaches inspired by Belief Revision have been proposed. Qi and Yang (2008) give an overview of approaches for Ontology Revision issued from the field of Belief Revision. The goal of methods in this field consists in defining a logic operator (a so-called revision operator) which ensures (in the case of addition of a new set of axioms) that the new information is part of the new ontology, that the ontology is consistent, that as little as possible is changed in the original ontology and that the obtained result is independent of the syntactic formalization of the added knowledge. The result of the application of the revision operator is a set of ontologies, in which each ontology is consistent and incorporates the newly added information. In this spirit, (Konstantinidis et al., 2008) have proposed a general framework which is able to deal automatically with any kind of change in the case of an RDFS ontology using a belief revision approach. The drawback of Ontology Revision approaches in many scenarios is that their goals are pursued in a domain-independent way. While the revision operator may propose the removal of a single axiom, it may be more pragmatic to perform a bigger change, i.e. to change more of the existing information, or to even refuse to integrate parts of the new knowledge.

Calvanese et al. (2010) have studied the applicability of belief revision approaches developed in the context of OWL-lite knowledge bases and have shown that most existing approaches lead to counter-intuitive results or to inexpressive results in this scenario. They therefore propose a new semantices for OWL-lite which allows for unique results when used for ABox evolution.

While Belief Revision approaches provide a fully automatic approach for integrating new axioms in the ontology, it might be desirable to let a domain engineer decide whether all the new axioms should be added to the ontology and which axioms to remove from the original ontology. This is for example useful when the newly acquired axioms are of possibly low quality, e.g. because they were acquired using some ontology learning tool. Nikitina et al. (2011) have proposed an interactive approach for this problem. They propose a method which combines automatic and manual evaluation of axioms while requiring as little input from the domain engineer as possible. The general idea is to define an impact function for each axiom which expresses how many axioms can be evaluated automatically if the axiom is accepted resp. refuted. Those axioms

with the highest impact are then evaluated first.

4.2.5 Ontology Amendment

The problem of automatically amending an ontology with new knowledge, i.e. of acquiring and integrating new knowledge in cases where the current ontology is not sufficient for solving the task at hand has been studied in the multi-agent systems community. They have researched how agents may automatically extend their ontologies in order to automatically adapt to new requirements. Their approaches discuss how agents can automatically integrate their ontologies with those of other agents to be able to communicate and collaborate with them.

Soh (2002) have proposed that agents should have their own vocabularies and that communication between them might be achieved through mapping tables which allow for translating from the own vocabulary to other agents' vocabularies. This requires that the meaning of a concept to which other agents map their vocabulary may not change over time as otherwise the mapping tables are invalidated. Another problem is posed by finding mappings between concepts of different agents. Bailin and Truszkowski (2002) have proposed a protocol which enables agents not only to identify communication problems, i.e. concepts and terms they do not understand, but also to acquire knowledge about the communication partners domain model. Based on the identified problems a dialogue between the agents is established which enables the agents to learn about the terms they could not understand (properly) before.

For the actual exchange of information about the different concepts and to gain an understanding of the communication partners' concept, Afsharchi et al. (2006) propose to query the other agents for positive and negative examples for the newly acquired concepts. The thus obtained examples may serve as input for a concept learning method which then returns a description of the new concept. This approach allows for acquiring the description of a concept of another agent in terms of the own domain conceptualization.

The cost of learning concepts has been taken into account by Packer et al. (2010b). They propose a method for selectively learning specific concepts thereby considering the cost of increasing the size of the ontology and of working with a bigger ontology. In contrast to previous approaches, their approach considers learning several concepts at a time instead of triggering the acquisition for every single concept.

4.2.6 Ontology Reduction

With the growth of an ontology in size, it becomes less efficient to use and query response times are increasing. In order to overcome this problem, *forgetting approaches* have been proposed. Their idea is to reduce an ontology in size by

deleting parts which are infrequently used or which are cheap to relearn (Packer et al., 2010a). To achieve this, a concept forgetting value is assigned to each concept which determines the importance of the concept for the ontology in terms of frequency of use and cost of acquisition. The actual forgetting then removes the concepts from the ontology. A challenge in this step consists in retaining all the knowledge which is not part of the forgotten concept, such as subsumption relations (Wang et al., 2008). While forgetting is feasible in lightweight ontology languages such as DL-lite (Wang et al., 2008) and $\mathcal{EL}$ (Konev et al., 2009), there may not be a valid knowledge base with a specific concept removed in case of more expressive ontolgoy languages (Wang et al., 2009). The problems of these approaches consist in their unguided operation which means that the forgetting algorithms may choose to remove concepts from an ontology that are needed in the future while other concepts may have become superfluous. However, these approaches are especially developed for the case of automatic ontology evolution where new knowledge is acquired automatically and thus also the reduction has to happen in a completely automatic way.

The approaches for forgetting from ontologies reduce the amount of knowledge which is represented in the ontology and thus make the ontology less knowledgeable with respect to some parts of the domain. This means that the new ontology is not able to answer all the queries that could be answered using the old ontology. Grimm and Wissmann (2011) have proposed an approach for reducing the size of the ontology without removing any knowledge from it. They aim at identifying and removing any parts of the ontology which are redundant, i.e. which could be removed and still be inferred using a reasoner afterwards.

4.3 Discussion

The last section shows that a considerable amount of work has been invested in developing methods which support the Ontology Change process. The approaches to support the ontology change process are manifold and various approaches have been developed to support the different kinds of changes that might be done on an ontology.

The refinement scenario is mainly supported with statistic approaches which learn parts of the ontology from various data sources. As external sources, mainly text corpora have been considered. The approaches rely on the application of text mining and information extraction methods. Based on the analysis of textual data with these means, rules and descriptions are derived. An interesting approach consists in using the current conceptualization of the domain to derive additional information which can not be obtained through deductive approaches. Two main classes of approaches may de distinguished in this class of approaches, those based on adaptations of ILP and those adapting classi-

cal machine learning approaches. However, the existing machine-learning based approaches have a two-fold shortcoming: either they rely on a manual feature definition, making them difficult to apply in practice, or they use reasoning for calculating the feature representation making their application expensive in many practical cases. We therefore think that learning approaches using a generally defined feature space which can be efficiently computed and do not rely on the solution of expensive reasoning tasks are an interesting alternative (see Chapters 5 and 6).

The knowledge obtained using these approaches can then be integrated into the ontology either using one of the fully automatic approaches, or an interactive revision approach may be used. The automatic approaches rely on automatic resolution of the problems arising from the change process. This requires expressive formalisms in which problems manifest themselves e.g. through inconsistency or incoherence. The alternative are interactive methods which identify the valid knowledge in collaboration with the user who either has to accept or refute certain axioms manually or who has to give examples which allow for accepting or refuting certain knowledge.

While the knowledge integration approaches may also be applied in other change scenarios, the requirements for knowledge acquisition change in these scenarios as the discrepancies between the domain model and the requirements resp. the domain drive these needs for changing the ontology. For the evolution scenario, approaches for extending the ontology with parts from other ontologies have been proposed as well as approaches for automatically eliminating superfluous knowledge once it is not needed anymore.

In the update scenario the case is slightly different. There, the adaptation of the knowledge has to be done based on information from the domain. Ontology learning approaches may help in acquiring information which is new in the domain. However, it is unclear which of the knowledge which is currently modelled is invalidated by the new information. Either interactive revision or completion approaches may be used in this case. Therefore, an approach which is able to identify the implications of new knowledge on the currently modelled information is needed. We propose to predefine the effects of domain updates using so-called update specifications (see 7).

Chapter 5

Kernel functions for RDF data

Incompleteness is an important problem in the context of knowledge bases. Even if certain information is required by the ontology, it cannot be guaranteed that this information is available.

To overcome this problem of missing data, methods for inducing information about entities are needed. While in some cases, reasoning may help to deduce the wanted information, there are situations which cannot be resolved by pure deductive methods. For example, we may require in an ontology that a person has an age attribute attached. This allows to infer for an entity of type person that it has an age, but the value of the age attribute remains unknown.

We propose to use machine learning methods to infer this missing information. The idea is to learn classification respectively regression models which can infer the missing property values. E.g., in the case of the missing age attribute, the solution would be to use the entities for which an age attribute is available to learn a regression model which is able to predict for entities not used in the training set what their age value should be.

Ontology learning methods and annotation tools use this kind of approach for extracting information from sources external to the ontology, like a text corpus. In this work, the goal is to infer the information from data which is internal to the ontology and the knowledge base, i.e. to use the available semantic data to infer new semantic data, which may then be integrated into the knowledge base.

This kind of approach requires methods for using semantic data as input for machine learning algorithms. The challenge thus consists in adapting existing machine learning algorithms which typically rely on vectors as data representation such that they can be used for mining semantic data. The essential design decision thus lies in defining suitable features to represent the entities at hand.

In this chapter we focus on the question how established machine learning algorithms can be made amenable to work on instances represented by means of RDF graph structures. We have decided to focus on graph structures as source for features as extracting semantic features such as class memberships is expensive. An additional reason for this choice ist the restricted deductive power of RDF reasoning, i.e. only few semantic properties which are not explicitely stated may be inferred.

Existing approaches for mining from semantic data have focused either on one specific learning problem (Rettinger et al., 2009; Huang et al., 2010; Bicer et al., 2011), i.e. the approach is not easily transferable to other learning problems, or they rely on clean, strongly axiomatised ontologies (Fanizzi and d'Amato, 2006; Fanizzi et al., 2008a,b) or features which have to be defined manually for each dataset (Bloehdorn and Sure, 2007). In our work we aim at an approach which can be applied to a wide set of learning problems on a wide set of ontologies without the need for extensive customisation.

Motivated by the broad applicability of graphs for modeling data, graph mining has received considerable interest over the last years and there has recently been significant progress in mining graph-structured data representations. A particularly successful research direction along this line is the use of *kernel methods* (Shawe-Taylor and Christianini, 2004).

Kernel methods provide a powerful framework for decoupling the data representation from the learning task: Specific *kernel functions* can be deployed depending on the format of the input data and combined with readily available *kernel machines* for supervised and unsupervised learning tasks, such as classification, regression, one-class classification or clustering. The challenge of learning from RDF data within this framework can be reformulated as designing adequate kernel functions for this representation.

In fact, various kernel functions for graph structures have been proposed over the last years, typically in the context of mining biochemical structures. However, graphs representing e.g. chemical structures have different properties than RDF: Chemical compounds usually have few node labels which occur frequently in the graph and nodes in these graphs have a low degree. In contrast, RDF node labels are used as identifiers occurring only once per graph and nodes may have a high degree.

Here, we investigate these issues, i.e. review the problems that arise when existing graph kernels are used with RDF graphs. Improving on that, we then introduce two versatile families of graph kernels based on intersection graphs and intersection trees, discuss why they can exploit the inherent properties of RDF better, and show that their computation can be performed efficiently.

The remainder of this chapter is organized as follows: Section 5.1, shortly review a number of existing graph kernels. In Section 5.2 we motivate and

describe the ideas underlying the kernel functions we have defined, Section 5.3 describes how instances are defined in the context of our approach, Sections 5.4 and 5.5 introduce two new families of kernel functions based on intersection graphs and intersection trees, discuss their properties and, specifically, their applicability for learning from RDF data. In Section 5.6, we report on several experiments which demonstrate the flexibility of the new kernel functions and evaluate their performance in practical settings compared to non-RDF-specific graph-kernels. We review related work on kernel functions for "semantic" data in Section 5.7 and conclude with a discussion and an outlook in Section 5.8.

5.1 Graph Kernels

As we only focus on the structure of the RDF graph for mining it, it would in principle be possible to apply any graph kernel to the problem at hand. We will here review some of the existing graph kernels with respect to whether they offer suitable representations of RDF graph structures.

Due to the expressivity of graph structures, the definition of kernel functions for arbitrary graphs has proven to be difficult. Research on kernel functions for structured data has thus concentrated for a long time on specific types of graphs with specific restrictions as e.g. *tree kernels* (see Shawe-Taylor and Christianini (2004) and references therein). In the spirit of the *Convolution Kernel* by Haussler (1999), which represents a generic way of defining kernels, kernel functions for general graphs have later been devised by counting common subgraphs of two graphs.

However, as the subgraph isomorphism problem, i.e. the problem of identifying whether a given input graph contains another input graph as its subgraph, is known to be NP-complete, it is not feasible to search for general subgraphs in the input graph. However, the search for common subgraphs with specific properties can often be performed more efficiently.

Horváth et al. (2004) have defined a kernel which is based on counting common cyclic and tree-like patterns in undirected labeled graphs, independent of their frequency. The kernel is defined on undirected graphs and thus not directly applicable to RDF as the direction of the application of an RDF property is an important property.

Shervashidze and Borgwardt (2009) propose a kernel that is based on counting common subtree-patterns of the input graphs. Here, only complete tree matches are counted. In their algorithm, a growing neighborhood of each node is encoded into the node labels. Two nodes are matching if they have the same label. In our experiments we have experienced that this kernel often only yields single matching nodes in the case of RDF graphs. No bigger structures can be matched. This is due to the property of RDF that each node has a unique

label: a tree pattern can only be matched, if the root nodes have all properties in common and if they have the same property value for each of the properties.

Gärtner et al. (2003) have defined a kernel which is based on counting walks within the graph. The feature space consists of one feature for each possible label sequence. The feature weight for each sequence is the number of times this sequence occurs in the graph, weighted by a factor for sequences of different lengths. The kernel which is obtained using this feature mapping can efficiently be computed based on the product graph: It is sufficient to calculate the limit of certain matrix power series using the adjacency matrix of the *product graph* of the two input graphs. However, the kernel value is obtained via calculating a matrix inverse which has cubic time complexity and becomes expensive for larger graphs. Section 5.2 will build on some of the ideas of this kernel.

5.2 Kernel Functions for RDF Graphs – Basic Ideas

In this section we present the core contribution of this chapter, two classes of kernel functions based on *intersection graphs* and *intersection trees*, specifically tailored to the properties of RDF.

Before introducing our kernel functions, we define the graph structure on which machine learning shall be carried out. The basic building blocks of RDF knowledge bases are triples (s, p, o), which are interpreted by s has a relationship of type p to o. A set of such triples can be represented as a graph:

Definition 31 (RDF Graph) *An RDF graph is defined by a set of triples of the form $G = \{(s, p, o)\} = (V, E)$, where the subject $s \in V$ is an entity, the predicate p denotes a property, and the object $o \in V$ is either another entity or, in case of a relation whose values are data-typed, a literal. The vertices $v \in V$ of G are defined by all elements that occur either as subject or object of a triple. Each edge $e \in E$ in the graph is defined by a triple (s, p, o): the edge goes from s to o and has label p.*

Note that this defines a multigraph which allows for multiple edges between the same two nodes, as long as the type of the relation is different.

The more advanced modelling elements of RDF can also be transformed into graph structures through the direct translation of the triples that represent them. However, the semantics of structures such as lists or reification is not represented specifically in the RDF graph.

We look at RDF entities as the instances for learning. For example, two sets of entities, identified by their URIs could be positive and negative classes in a classification scenario. The argument entities' neighborhood in the overall RDF graph forms the basis for their kernel-induced feature representations.

Essentially, all proposed kernel functions are thus based on a neighborhood graph which is obtained by a breadth-first search *up to depth k* starting from the entity of interest. We have defined two versions of the neighborhood graph: either all encounters of a node label are mapped to the same node in the neighborhood graph or each encounter of a node label defines a separate node in the neighborhood graph . Here, each encounter of a node is defined by a walk of length $\leq k$ from the entity of interest to the node. In the first case we call the neighborhood graph *instance graph*, in the second case we call it *instance tree*. A more detailed explanation of instance graphs and instance trees can be found in the following sections.

We define RDF kernels in a similar manner to other graph kernels by adopting the idea of counting subgraphs with a specific structure in the input graphs. Formally, this means that the (implicit) feature mapping for a neighborhood graph G representing the instance given a set of feature graphs $\mathcal{H}$ should be of the form:

$$\phi(G) = \sum_{h \in \mathcal{H}} |\{g | g \subset G \wedge h \cong g\}| \tag{5.1}$$

The essential difference is that, as RDF builds on unique node labels, each RDF subgraph h can occur at most once in the input graph. This is not the case in general graphs, where it is common that several nodes carry the same label – thus yielding potentially several equivalent subgraphs.

Blank nodes seem to present a special case at first: blank nodes are nodes without labels and thus blank nodes are not identified by their URI. Thus, blank nodes seem to break the unique label assumption which is a basis for the definition of our kernel functions. However, a blank node is given a temporary label in the serialisation of an RDF graph. We treat these labels as normal labels which allow for the identification of the blank node and thus the distinction between labeled nodes and blank nodes is not necessary in our approach.

Therefore, when calculating the kernel function between two RDF graphs, it is not necessary to identify the interesting structures and their frequencies in the two graphs separately. Instead, it is sufficient to analyze a single structure which contains the features of interest *common* in both input graphs. Gärtner et al. (2003) have proposed kernel functions which are based on counting common structures in the *direct product graph*. In the case of graphs with unique node labels, like RDF, this is equivalent to what we call the *Intersection Graph* which we define in Section 5.4.

For each of the definitions of the neighborhood graphs sketched above, we have defined a way of representing their *common* structures , which are used as basis for the two families of kernel functions we define: In Section 5.4 we will present the first type of kernel functions which are based on *intersection graphs*

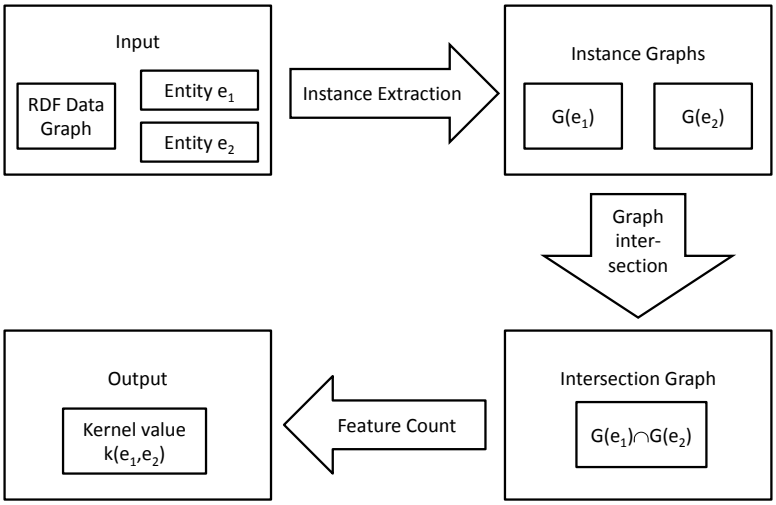

Figure 5.1: Process of kernel calculation

(obtained from two instance graphs), in Section 5.5 the second type which is based on *intersection trees* (on the basis of instance trees) will be presented.

The common structures that have been extracted are the structures which are used for counting the common features of the two entities. The features have to be chosen in a way that the kernel property of the resulting kernel function can be ensured. In most cases we will argue the validity of the proposed kernel functions through an explicit mapping ϕ into a feature space where the dot product gives the same result as counting the common structures.

An overview of the process of calculating the proposed kernel functions is given in Figure 5.1. First, the neighborhood graphs of the two input entities are extracted. Then, the intersection of the two instance graphs is built which represents the common elements of the two instance graph. In the last step, certain features are counted in this intersection graph. In the following we will present details on how these steps may be instantiated.

5.3 Instance extraction

We assume that the entities in the dataset are all represented in a single data graph. Note that if there are several input graphs, these can easily be merged into a single graph. However, the data should be modelled using the same vocabularies, as these provide the identifiers and labels we use for checking whether instances have common features.

Given the data graph and an instance, the first step consists in identifying the part of the graph which is relevant for the entity at hand. We decided to have extraction methods which are domain-independent and that do not require any knowledge on what kind of information is modelled. We assume that all information relevant for the entity of interest is part of its neighbourhood and has a distance to the entity which is less or equal to some maximum distance d.

Note that it would also be possible to explicitely model which kind of relations and information are to be considered in the instance description. However, this would require a lot of manual effort, as for each classification problem a decision which information in the entities' neighborhood is relevant for the classification has to be made.

When considering the d hop neighborhood, this consideration is still necessary with respect to the choice of d. If d is set to 1, only the direct properties of the entity are considered. However, it may be useful to use information which is farther away from the entity. Consider for example a setting where the entities of interest are people. An entity directly related to the person is the city they come from, which itself is connected to the country which it belongs to. Two people from the same country may however be more similar than people from different countries. Using indirect properties accounts for this source of similarity. The same kind of reasoning may be employed to argue for the use of larger ds. However, with the growth of d the extracted graphs grow exponentially (and thus also the cost of kernel calculation) and also noise which is not helpful for distinguishing between the entities of interest is introduced. If d is chosen too large, the whole data graph may be equivalent to each instance graph and no distinction between the entities is possible. It is thus crucial to choose d appropriately.

We have defined two different kinds of graphs for representing instances: instance graphs and instance trees. While the first type represents a subgraph of the neighborhood graph, the second one is a tree containing the structure of the entities' neighborhood. The motivation for not only using the neighborhood graph is that trees can often be handled more efficiently than general graphs.

5.3.1 Instance Graphs and Intersection Graphs

An instance graph of depth d can be extracted from the data graph using breadth-first search up to depth d starting from the entity of interest e. All elements encountered during the search are added to the instance graph.

The instance graph of depth d for entity e can be obtained using Algorithm 2.

Given the two instance graphs, the next step consists in extracting the parts which the two graphs share. The intersection graph of two graphs is a graph containing all the elements the two graphs have in common.

Algorithm 2: Extraction of an instance graph of depth d for entities e

Input: entity e
maximum graph depth d
Data: RDF data graph $G = (V, E)$, where labels of entities e
Result: *graph*: Instance graph of depth d for entity e

```
1  graphNodes ← {e}
2  graphEdges ← ∅
3  newNodes ← {e}
4  for i = 1 : d do
5  |  nodes ← newNodes;
6  |  for node ∈ nodes do
7  |  |  newNodes ← {e_i|(node.label, p, e_i) ∈ G}
8  |  |  newEdges ← {(node.label, p, e_i)|(node.label, p, e_i) ∈ G}
9  |  |  graphNodes ← graphNodes ∪ newNodes
10 |  |  graphEdges ← graphEdges ∪ newEdges
11 |  end
12 end
13 graph ← (graphNodes, graphEdges)
```

Definition 32 (Intersection Graph) *The intersection graph $G_1 \cap G_2$ of two graphs G_1 and G_2 is defined as:*

$$\begin{aligned} V(G_1 \cap G_2) &= V_1 \cap V_2 \\ E(G_1 \cap G_2) &= \{(v_1, p, v_2) | (v_1, p, v_2) \in E_1 \wedge (v_1, p, v_2) \in E_2\} \end{aligned}$$

To illustrate the notion of instance graphs and intersection graphs consider the Example in Figure 5.2. In the given graph, all nodes are reachable within 2 steps from the node `person100`. Thus, the instance graph of depth 2 for the entity `person100` corresponds to the whole datagraph. The instance graph for `person200` is depicted in Figure 5.3. Any node whose distance from `person200` is bigger than 2 is not part of the instance graph. Also all relations whose subject is more than 1 hop away from `person200` are not part of the instance graph. The intersection graph for `person100` and `person200` is obtained by intersecting the two instance graphs. As in this case any element which is part of the instance graph for `person200` is also part of the instance graph for `person100`, the instance graph for `person200` is equivalent with the intersection graph for the two entities.

Note that if the intersection graph contains a given subgraph, then this subgraph is also a subgraph of the two input graphs. Inversely, if a subgraph is contained in both instance graphs, it is also part of the intersection graph. Now recall that a kernel function is defined as the scalar product in some feature

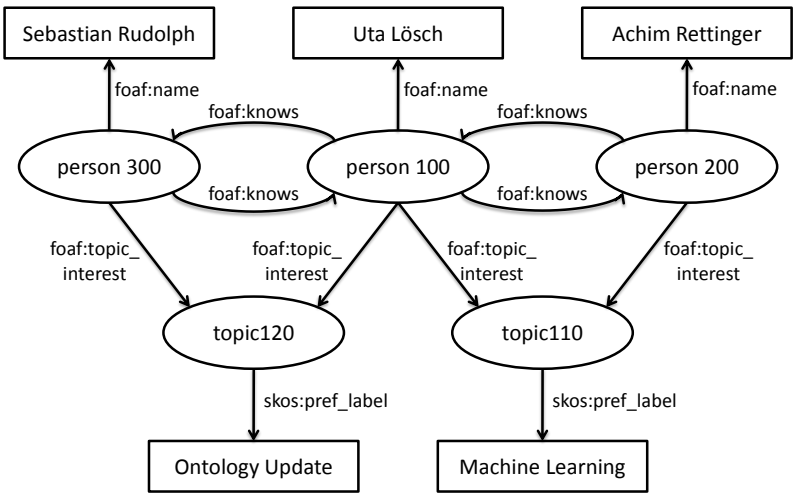

Figure 5.2: Example for instance graphs: The instance graph of depth 2 for `person100` corresponds to the whole data graph in this case.

space. In order for the kernel function to be a valid scalar product, the corresponding feature space has to be defined. In the case of the instance graphs, any structure which is present in both input graphs is also part of the intersection graph: thus, counting the features (which can occur at most once due to the unique names in RDF) in the instance graphs and multiplying the resulting feature vectors leads to the same result as counting the structures in the intersection graph.

Thus, calculating a kernel function based on a feature mapping as defined in Equation 5.1, i.e. which is based on counting certain subgraphs in the instance graph, can be reduced to constructing the intersection graph in the first step and then counting the substructures of interest therein. This is for example possible for walks, paths, cycles, or connected subgraphs. The corresponding kernel functions will be defined in Section 5.4.

5.3.2 Instance Trees and Intersection Trees

The use of the intersection graph may become problematic as its calculation is potentially expensive: the whole instance graph for each entity has to be extracted and the two graphs have to be intersected explicitely.

However, the size of the instance graph grows exponentially with the number of hops which are crawled from the entity. Merging the two steps of extracting the instance graphs and intersecting them is not directly feasible: Consider an entity e which can be reached within k hops from both entities of interest e_1 and e_2, but through different paths. In this case, e would be part of the

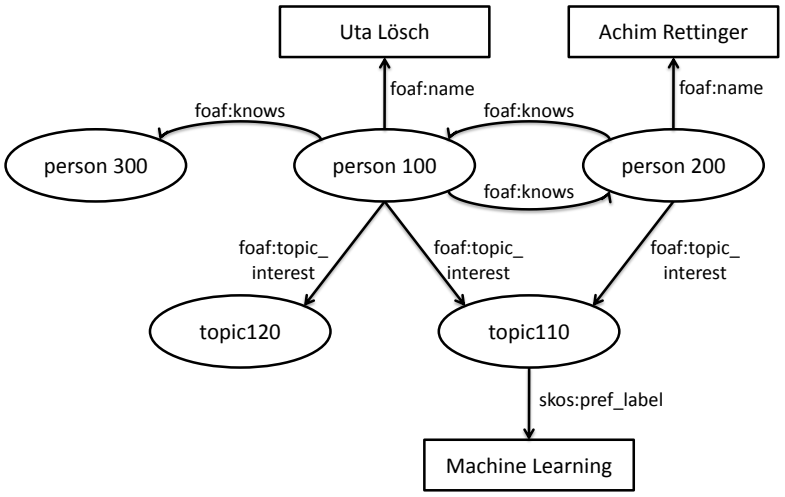

Figure 5.3: Example for instance graphs: The instance graph of depth 2 for `person200` based on the data graph in Figure 5.2.

intersection graph, but it would not be reachable from either e_1 or e_2 in this graph. In the following section, we present a different way of extracting common neighborhoods of two entities e_1 and e_2, which enables a direct construction of the common properties, without building the instance graphs first. This alternative method is based on the use of instance trees instead of instance graphs. To obtain instance trees as neighborhood descriptions, we use a method based on the graph expansion with respect to an entity of interest e (as for example defined in Güting (1992)).

Definition 33 (Graph Expansion) *The expansion $X(e)$ of a graph G with respect to entity e is a tree defined as follows:*

- *If e does not have any successors, then $X(e)$ is the tree consisting only of the node e.*
- *If $v_1, \ldots, v_n$ are the successors of e, then $X(e)$ is the tree $(e, X(v_1), \ldots, X(v_n))$ (in prefix notation).*

In principle, the graph expansion could grow infinitely (if the graph contains cycles). To avoid this problem and to limit the size of the obtained trees the graph expansion is bound by some maximal depth k. While in the original RDF graph, node labels were used as identifiers, i.e. each node label occured exactly once, this is not true in the expanded graph. If there is more than one path from an entity e to another entity e', then e' will occur more than once, and thus the label of e' is not unique anymore.

An intersection of the instance trees leads to an intersection tree in a similar spirit as the intersection graph presented in Section 5.3.1. We introduce two changes to the instance trees as obtained by direct expansion from the data graph: first, each occurence of either entity of interest e_1 or e_2 is replaced by a common dummy node label which does not occur elsewhere in the graph, and second, when calculating the intersection graph, only parts which are connected to the root element in the intersection graph are retained. The structure which we obtain by these steps is called the intersection tree.

The procedure for obtaining an intersection tree as we have described it above, yields the same problematic overhead as does the construction of the intersection graph. However, it turns out that the intersection tree can be extracted directly from the data graph without constructing the instance trees explicitely. This is especially due to the fact that all elements of the intersection tree are connected to the root of the tree. Thus, the construction of an intersection tree can be done much more efficiently than the construction of the corresponding intersection graph.

The construction of an intersection tree is illustrated in Algorithm 3. The basic idea of the algorithm is to extract the intersection tree $it_d(e_1, e_2)$ directly from the data graph. Starting from the two entities e_1 and e_2 the intersection graph is built using breadth-first search. Two cases have to be distinguished. In cases where one of the entities e_1 or e_2 is found a new node is added to the tree with a dummy label. For nodes with this label, the common relations of e_1 and e_2 are added as children. The second case are all nodes which do not correspond to one of the entities: for them, a new node with the node's URI resp. label is added to the tree, the children of these latter nodes are all relations of this node in the data graph.

To illustrate the idea of instance trees and intersection trees, we continue the running example and show the instance trees of depth 2 for the entities `person100` and `person200` in Figures 5.4 and 5.5. The corresponding intersection tree is shown in Figure 5.6. Its root node is a dummy node with a newly introduced label `source` which is used to replace any occurrence of `person100` and `person200` in the tree. The children of the `source` node are all properties which `person100` and `person200` have in common (see line 7 in Algorithm 3), in this case `topic110`. Additionally, `person100` is linked to `person200` via the `foaf:knows` property and vice versa, thus an additional node `source` is introduced. In the second step, all children of the `topic110` are added to the intersection trees. The children of the second `source` node are obtained using the same procedure as in the first iteration.

Algorithm 3: Extraction of an intersection tree of depth d for entities e_1 and e_2

Input: entities e_1, e_2
maximum tree depth d
Data: RDF data graph $G = (V, E)$, where labels of entities e_1 and e_2 are replaced by source
Result: *tree*: Graph expansion $X(e_1 \cap e_2)$ of depth d

```
1  tree ← new Node("source",0)
2  newLeaves ← {tree}
3  for i = 1 : d − 1 do
4      leaves ← newLeaves;
5      for leaf ∈ leaves do
6          if leaf.label="source" then
7              ce ← {e_i|(e_1, p, e_i) ∈ G ∧ (e_2, p, e_i) ∈ G}
8              for p : (e_1, p, e_2) ∈ G ∧ (e_2, p, e_1) ∈ G do
9                  ce.add("source")
10             end
11         end
12         else
13             ce ← {e_i|(leaf.label, p, e_i) ∈ G}
14         end
15         for c ∈ ce do
16             if c ∈ {e_1, e_2} then
17                 label="source"
18             end
19             else
20                 label=c.uri
21             end
22             child ← new Node(label, leaf.depth + 1)
23             leaf.addChild(child)
24             newLeaves.add(child)
25         end
26     end
27 end
```

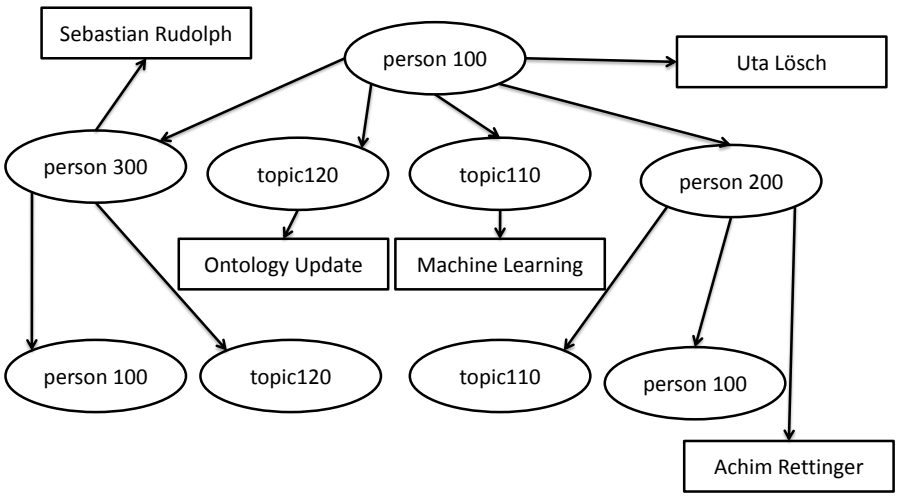

Figure 5.4: Instance tree of depth 2 for `person100`

5.4 Kernel Functions Based on Intersection Graphs

Once the intersection graph is calculated, the common features of the two graphs can be counted in the intersection graph. We have defined several kernel functions which are based on different sets of features: subgraphs, connected subgraphs, walks and paths.

5.4.1 Edge-Induced Subgraph Kernel

The set of edge-induced subgraphs qualifies as a candidate feature set.

Definition 34 (Edge-Induced Subgraphs) *An edge-induced subgraph of* $G = (V, E)$ *is defined as* $G' = (V', E')$ *with*

$$\begin{aligned} E' &\subseteq E \\ V' &= \{v \mid \exists u, p : (u, p, v) \in E' \vee (v, p, u) \in E'\} \end{aligned}$$

We denote the edge-induced subgraph relation by $G' \subseteq G$.

Now recall that a graph $G = (V, E)$ has $2^{|E|}$ edge-induced subgraphs (as all subsets of edges define an edge-induced subgraph). Thus we define the subgraph kernel by:

Definition 35 (Subgraph Kernel) *The subgraph kernel is defined as:*

$$\kappa_{subgraph}(G_1, G_2) = 2^{|E(G_1 \cap G_2)|}$$

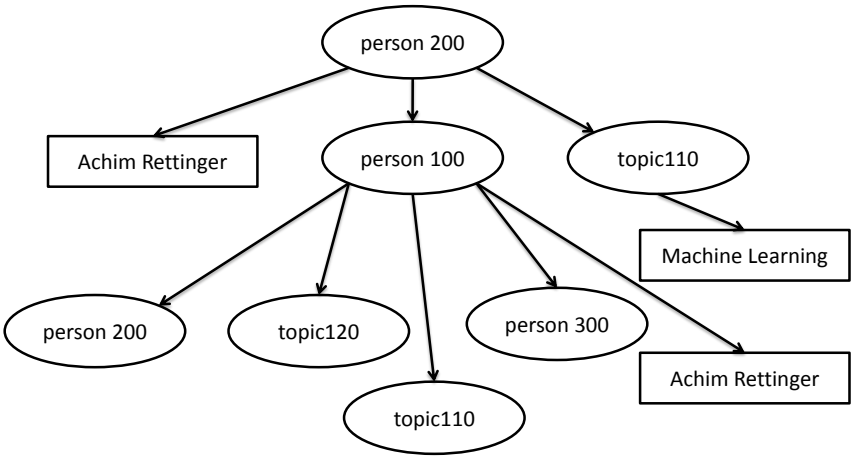

Figure 5.5: Instance tree of depth 2 for `person200`

The corresponding feature mapping is:

$$\phi_g(G) = \begin{cases} 1 & g \subseteq G \\ 0 & otherwise \end{cases}$$

where $g \in \mathcal{G}$ and $\mathcal{G}$ is the set of valid RDF graphs.

The subgraph kernel is a valid kernel function, as counting edge-induced subgraphs in the intersection graph is equivalent to performing the feature mapping explicitly and calculating the dot product of the two feature vectors. The dot product of the two feature vectors corresponds to counting elements which are part of both input graphs (as subgraphs occur at most once in an RDF graph, features occur at most once). The edges that are present in both inputs are by construction the same as those which are part of the intersection graphs.

5.4.2 Connected Subgraphs

Connected elements within the intersection graphs are probable to yield more interesting results than a set of arbitrary relations taken from the intersection graph. We have therefore defined additional kernels whose features are restricted to subsets of all edge-induced subgraphs.

A subgraph is called *connected* if there exists a semi-walk between each pair of nodes from the graph, i.e. $\forall u, v \in V' \exists u = v_0, v_1, \ldots, v = v_{n+1}$ with $v_1, \ldots, v_n \in V'$ and $\forall (v_i, v_{i+1}) \exists p : (v_i, p, v_{i+1}) \in E' \vee (v_{i+1}, p, v_i) \in E'$.

Definition 36 (Edge-induced Connected Subgraphs Kernel) *We define*

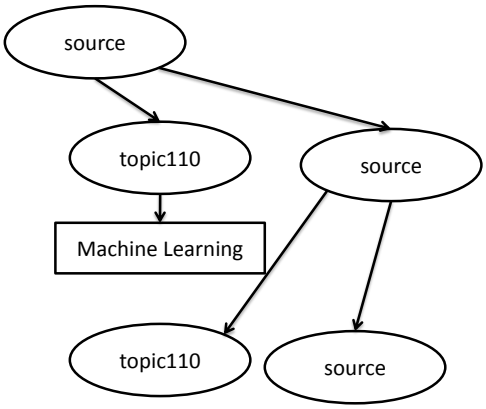

Figure 5.6: Intersection tree of depth 2 for `person100` and `person200`

the Edge-induced Connected Subgraphs Kernel *is defined as:*

$$\kappa_{csg,k}(G_1, G_2) = |\{g | g \subset G, |E(g)| \leq k, g connected\}|$$

The corresponding feature mapping is:

$$\phi_g(G) = \begin{cases} 1 & g \subseteq G \\ 0 & otherwise \end{cases}$$

where $g \in \mathcal{G}_{conn}$ *and* $\mathcal{G}_{conn}$ *is the set of connected RDF graphs.*

There is no general formula for the number of connected edge-induced subgraphs. Thus, an algorithm for counting these elements is needed. Algorithm 4 counts all connected elements up to size k in a graph. The algorithm starts by identifying all subgraphs with one edge. In each iteration the size of the subgraphs is increased by one and an edge which is not yet part of the graph is added to it. Bigger subgraphs are identified by adding one of the edges in the current subgraphs neighborhood to the subgraphs found in the last iteration. A weight factor λ has been introduced which allows to give more or less weight to bigger subgraphs.

As the algorithm works inductively, it is sufficient to change the termination criterion in order to count all connected elements.

5.4.3 Walks and Paths

Counting connected subgraphs is expensive in practical, as the check whether a specific graph has already been found, requires comparing graphs for equality.

Algorithm 4: Counting connected edge-induced subgraphs up to size k

Input: Graph $G = (V, E)$,
maximum graph size k,
weight λ>0
Result: κ

$S_1 = E$
$\kappa = \lambda|S_1|$
for $i = 2$ *to* k **do**
  $S_i = \emptyset$
  for $s \in S_{i-1}$ **do**
    $candidates = \{(u, p, v) \in E \backslash s : u \in V(s) \vee v \in V(s)\}$
    for $c \in candidates$ **do**
      $S_i = S_i \cup (s \cup c)$
    end
  end
  $\kappa = \kappa + i\lambda|S_i|$
end
return κ

We are therefore interested in subgraphs, which can be counted more efficiently. We have focused on *walks* and *paths* as interesting subsets, as they represent property chains in RDF.

Definition 37 (Walk, Path) *A* walk *in a graph $G = (V, E)$ is defined as a sequence of vertices and edges $v_1, e_1, v_2, e_2, \ldots, v_n, e_n, v_{n+1}$ with $e_i = (v_i, p_i, v_i + 1) \in E$. The length of a walk denotes the number of edges it contains.*

A path *is a walk which does not contain any cycles i.e. a walk for which the additional condition $v_i \neq v_j \forall i \neq j$ holds. We denote the set of walks of length l in a graph G by $walks_l(G)$, the paths of length l by $paths_l(G)$.*

Definition 38 (Walk Kernel, Path Kernel) *The Walk Kernel for maximum path length l and discount factor $\lambda > 0$ is defined by:*

$$\kappa_{l,\lambda}(G_1, G_2) = \sum_{i=1}^{l} \lambda^i |\{w | w \in walks_i(G_1 \cap G_2)\}|$$

The Path Kernel is defined in an analogous manner:

$$\kappa_{l,\lambda}(G_1, G_2) = \sum_{i=1}^{l} \lambda^i |\{p | p \in paths_i(G_1 \cap G_2)\}|$$

The feature space of interest consists of one feature per walk w (resp. path):

Algorithm 5: Counting walks up to length l

Input: (Intersection) Graph $G = (V, E)$,
maximum walk length k
weight $\lambda > 0$ (for giving different weights to walks of different lengths)
Result: κ Count of walks up to length k weighed by weights w_i

1 $W_1 = E$
2 $\kappa = \lambda |W_1|$
3 **for** $i = 2$ *to* k **do**
4 $\quad W_i = \{(v_1, \ldots, v_{i-1}, v_i) | (v_1, \ldots, v_{i-1}) \in W_{i-1} \wedge \exists p : (v_{i-1}, p, v_i) \in E\}$
5 $\quad \kappa = \kappa + \lambda^i |W_i|$
6 **end**
7 **return** κ

$$\phi_w(G) = \begin{cases} \lambda^{length(w)} & w \in G \\ 0 & w \notin G \end{cases}$$

In the definition, the parameter $\lambda > 0$ serves as a discount factor and allows to weight longer walks (paths) different from shorter ones. If $\lambda > 1$ then longer walks (paths) receive more weight, in case of $\lambda < 0$ shorter ones contribute more weight.

As paths and walks are edge-induced substructures of a graph, the validity of the proposed kernel functions can be shown in the same way as that of the subgraph kernel. The kernel function can be calculated using Algorithm 7. : walks of length i are constructed by extending walks of length $i - 1$. In each iteration the walks found in the previous iteration are extended by appending an edge at the end of the walk. For counting paths, the condition that $v_i \notin \{v_1, \ldots, v_{i-1}\}$ has to be added in line 4.

Note that a different way of calculating these kernel functions is possible based on the powers of the intersection graph's adjacency matrix. The adjacency matrix M is a representation of a graph in the form of a matrix with one row and one column per node in the graph and entries $x_{ij} = 1$ if the graph contains an edge from node i to node j, 0 otherwise. Each entry x_{ij} of the k^{th} power of M can be interpreted as the number of walks of length k existing from node i to node j. Therefore, the number of walks up to length k in the graph can be obtained as $\sum_{i=1}^{k} \sum_{j=1}^{n} \sum_{l=1}^{n} (M^i)_{jl}$. By setting the elements x_{ii} of M^k to 0, this formula can also be used for the path kernel.

Gärtner et al. (2003) use this approach for counting walks up to infinite length by calculating the limes $k \to \infty$ of the matrix power series. They also use a weight factor to give different weight to walks of different length. However, for the matrix power series to converge, their weight factor has to be smaller

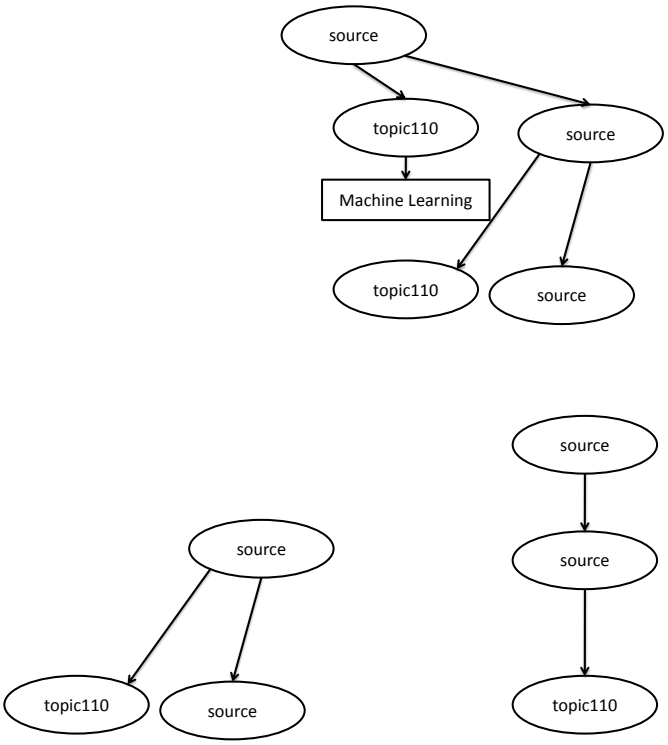

Figure 5.7: Example of a full (left) and a partial subtree (right) in the intersection tree (upper part)

than 1. Thus, in their kernel it is only possible to give smaller weight to larger structures. In the case of RDF, it may however be preferable to give more weight to larger structures as those convey more meaning.

5.5 Kernel Functions Based on Intersection Trees

As in the case of the intersection graphs, our proposed kernel functions are based on counting elements in the intersection trees. The features of interest are restricted to features which contain the root of the tree. This is because features which are not connected to the root element may be present in each instance tree, but may by construction not be part of the intersection tree.

5.5.1 Full Subtrees

The first kernel function we propose based on the intersection tree is the *full subtree kernel*, which counts the number of full subtrees of the intersection tree

$it_d(e_1, e_2)$, i.e. of the intersection tree of depth d for the two entities e_1 and e_2. A full subtree of a tree t rooted at a node v is the tree with root v and all descendants of v in t (see Figure 5.7 for an example).

Definition 39 (Full Subtree Kernel) *The* Full Subtree kernel *is defined as the number of full subtrees in the intersection tree. Subtrees of different height are weighted differently using a discount factor* λ.

$$\kappa_s t(e_1, e_2) = st(root(it_d(e_1, e_2)))$$

where

$$st(v) = 1 + \lambda \sum_{c \in children(v)} st(c)$$

The corresponding feature mapping consists of one feature per subtree:

$$\phi_s(x) = \begin{cases} \lambda^i & s \text{ subtree of } t \text{ with height } i \\ 0 & else \end{cases}$$

Counting the number of full subtrees in the kernel is equivalent to counting walks which start at the root of the intersection tree. This is the case because in a tree there is exactly one path from the root of the tree to every node in the tree and there is one full subtree per node in the tree. Thus, the two values are equivalent to the number of nodes in the tree. The full subtree kernel is a valid kernel function due to this equivalence.

5.5.2 Partial Subtrees

Given a tree $T = (V, E)$, its partial subtrees are defined by subsets $V' \subset V$ and $E' \subset E$ such that $T' = (V', E')$ is a tree. We propose to define a kernel function which counts the number of partial subtrees in the intersection tree $it_d(e_1, e_2)$ which are rooted at the root of $it_d(e_1, e_2)$.

Definition 40 (Partial Subtree Kernel) *The* Partial Subtree Kernel *is defined as the number of partial trees that the intersection tree contains. A discount factor* λ *gives more or less weight to trees with greater depth:*

$$\kappa_{pt}(e_1, e_2) = t(root(it_d(e_1, e_2)))$$

where t *is defined as:*

$$t(v) = \prod_{c \in children(v)} (\lambda t(c) + 1)$$

The function $t(v)$ *returns the number of partial subtrees with root* v *that the tree rooted at* v *contains weighted by depth with a discount factor* λ. *The corresponding feature mapping consists of one feature per partial subtree:*

The corresponding feature space consists of one feature per partial tree up to depth d where each occurrence of root e_i is replaced by a dummy node in the data graph. The value of each feature is the number of times a partial tree occurs.

Partial trees of limited breadth

An alternative to counting all partial trees which may be very broad we propose an alternative which consists in counting all partial trees up to some maximum breadth.

Definition 41 (Limited Breadth Partial Tree Kernel) *The Limited Breadth Partial Tree Kernel is obtained by counting all partial trees of the intersection graph whose maximum degree is smaller than b:*

$$\kappa(e_1, e_2) = t_b(root(it_d(e_1, e_2)))$$

where $t_b(v)$ is defined as:

$$t_b(v) = \sum_{i=1}^{min\{b,n\}} \sum_{s \in C_i} \prod_{c \in s} t(c) + \begin{cases} 1 & n \leq b \\ 0 & n > b \end{cases}$$

Here, n denotes the number of children of v and C_i the subsets of children(v) with i elements.

Intuitively, the formula reflects the following line of thought: Given a node v, one partial tree is obtained by using only this node. Further trees are obtained by considering each of the subtrees rooted at one of the children separately. In further steps more and more of the subtrees rooted at one of the children of v are combined to obtain further partial subtrees.

However, the cost of calculating this kernel function grows exponentially with growing depth. When changing the sum such that it counts subtrees of unlimited breadth, the partial tree kernel is obtained.

5.6 Evaluation

To validate our approach we implemented our kernel functions and evaluated them on two real world data set. In two scenarios, we compare our kernels to two kernels devised for general graphs: the Weisfeiler-Lehman kernel (Shervashidze and Borgwardt, 2009) and the Gärtner kernel (Gärtner et al., 2003). The implementation we provide is based on SVM*light* (Joachims, 1999) together with the JNI Kernel Extension.[1] The kernel functions are implemented in Java us-

[1] `http://people.aifb.kit.edu/sbl/software/jnikernel/`

ing JENA[2] for processing RDF data. The trade-off parameter c of the Support Vector Machine learning was set to 1 in all evaluation runs.

5.6.1 Evaluation procedure

Results are obtained using leave-one-out cross validation. The reported evaluation measures are error, precision, recall and F-measure. We will shortly explain the evaluation procedure and the evaluation measures in the following.

5.6.2 Cross-Validation

The goal of classifier evaluation is to estimate how well it performs on unknown data. Thus, the available labeled data is separated in a training set and a test set. While the classifier will be trained on the training set, the trained model will later on be used to score the test data. The classifications which are thus obtained are then compared to the labels of the test data to decide whether the model correctly classifies the data.

However, the evaluation measures obtained through a separate training and test set are heavily dependent on how the training and test split are chosen. A method for overcoming this problem is *n-fold cross-validation*. Here, the evaluation measures are obtained as average of the evaluation measures of n distinct models. For training these modes the available data is split in n parts - so called folds. For training each model $n-1$ of the folds are used as training data and the remaining fold is used as test data.

Typical procedures are 5-fold, 10-fold or *leave-one-out cross validation*. The latter is the extreme case where each test set consists of exactly one instance. With growing n the results better approximate the performance of a classifier trained on all training data, however, the complexity of the evaluation increases, as a model has to be trained for each fold. All results reported in the following were obtained using leave-one-out cross-validation.

5.6.3 Evaluation measures

Evaluation of classification is frequently done using the well known evaluation measures *accuracy*, *precision*, *recall* and *F1-measure*. The definition of these measures is based on binary classification scenarios where one of the target classes is denoted the positive class and one is denoted the negative class.

Definition 42 (Evaluation measures for Classification Systems) *Based on the types of errors that are defined in Table 5.1 different quality metrics can be defined:* Accuracy *denotes the fraction of documents that are correctly classified:*

$$acc = \frac{TP + TN}{TP + FP + FN + TN}$$

[2] `http://jena.sourceforge.net`

classified as\class	positive	negative
positive	true positive (TP)	false positive (FP)
negative	false negative (FN)	true negative (TN)

Table 5.1: Different types of errors in classification tasks

Recall *denotes the fraction of relevant documents that are actually classified as relevant:*

$$recall = \frac{TP}{TP + FN}$$

Precision *is defined as the fraction of documents that are actually relevant among all documents that are classified as relevant:*

$$precision = \frac{TP}{TP + FP}$$

As maximum precision is achieved at a minimum recall and vice versa, the harmonic mean of both measures is frequently considered. The measure is known as F1 measure.

$$F1 = \frac{2 * precision * recall}{precision + recall}$$

5.6.4 Evaluation on SWRC Ontology

In a first evaluation setting, we applied our kernels to the *person2affiliation* task presented by Bloehdorn and Sure (2007). We report their results although their approach is not purely based on graph kernels. Additionally, we compared to two kernels that have been deviced for general graphs: the Weisfeiler-Lehman kernel (Shervashidze and Borgwardt, 2009) and the Gaertner-kernel (Gärtner et al., 2003).

Data set

The evaluation uses data from the SWRC ontology (Sure et al., 2005) and the metadata which is available in the Semantic Portal of the institute AIFB. The ontology models key concepts within research communities, among them people, publications, projects and research topics. The evaluation data consists of 2,547 entities of which 1,058 can be derived to belong to the person class. 178 of these persons are affiliated with one of the research groups at AIFB, 78 of them being currently employed. Additionally, there are 1232 instances of type publication, 146 instances of type research topic and 146 instances of type project. The entities are connected by a total of 15,883 relations among them. Additionally, there are 8,705 datatype properties, i.e. properties linking an entity to a literal. The evaluation setting defined by Bloehdorn and Sure (2007) consists in classifying staff members with respect to their affiliation in

Instance depth	1	2	3
#Instances		178	
#Labels		8284	
Average #nodes	21.78	139.43	617.35
Max. #nodes	223	1451	3752
Average #edges	21.10	284.14	1546.94
Max. #edges	222	3900	13825

Table 5.2: Statistics of the instance graphs in the SWRC dataset

research groups at AIFB. All relations denoting the affiliation of a person with a research group were deleted from the training data. We report statistics about the instance graphs obtained in Table 5.2. Note that – as we would expect – the size of the instance graphs grows exponentially with the instance depth and that with increasing instance depth the edge-node ratio increases.

Compared approaches

We compare the results obtained using the kernel functions defined in Sect. 5.2 to the best configuration obtained in the original paper. This kernel configuration, denoted by *sim-ctpp-pc* combines the common class similarity kernel described in their paper with object property kernels for the *workedOnBy*, *worksAtProject* and *publication*. The Weisfeiler-Lehman kernel and the Gärtner kernel are directly comparing graphs. We applied these kernels to the instance graphs of depth 2 which were extracted from the RDF data graph. The maximum depth of trees in the Weisfeiler-Lehman kernel was set to 2, the discount factor for longer walks in the Gärtner kernel was set to 0.5. As additional baseline we defined the edge kernel which is obtained by counting the edges in the intersection graph. It can also be obtained by setting the maximum path/walk length to 1 in the path/walk kernel. Results are reported in Table 5.5, which can be found at the end of the chapter. We compared the performance of our kernels on the whole data set (including the schema) to a setting where all relations which are part of the schema were removed from the data (lowest part in Table 5.5).

Discussion

Our experiments show that it is not obvious what the best discount factors for the Partial Subtree kernel are, as results vary strongly. The other kernel functions we proposed perform more robustly. Another interesting outcome is the observation, that the exponential cost of increasing the instance depth (see Table 5.2) does not necessarily improve the results (see Fig. 5.8). This is probably because the additional data which becomes part of the data graph is not necessarily useful for distinguishing between entities. Overall, the results

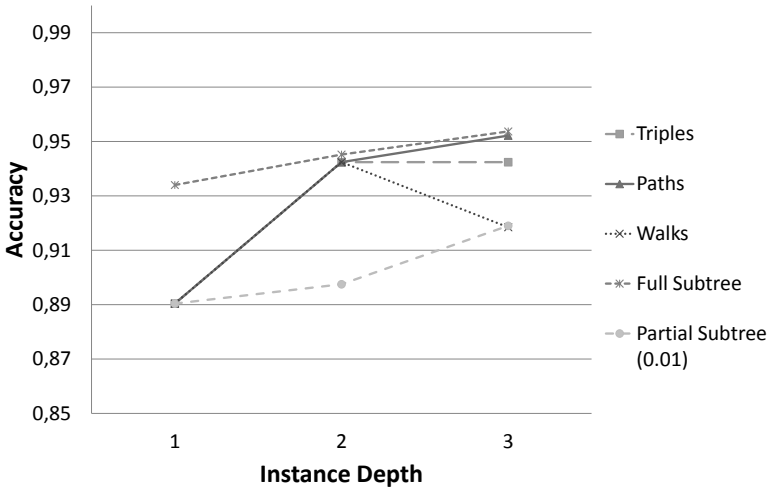

Figure 5.8: Accuracy for different kernels given instance depth

do not indicate that one of the kernels and one of the kernel configuration is superior to the others.

In addition, our results show that with specific parameters our kernels reach comparable error and higher F1-measure than the kernels proposed by Bloehdorn and Sure. Considering that our approach is generic and can be used off the shelf in many scenarios, while their kernel function was designed manually for the specific application scenario, this is a positive result. Our kernel functions also perform well with respect to other graph kernels: The Weisfeiler-Lehman kernel is not able to separate the training data in this case as it can match only very small structures. While the Gaertner kernel achieves results which are comparable to our results, its calculation is more expensive - due to the cubic time complexity of the matrix inversion. Last but not least, a surprising result is that the kernels which are based on the intersection graph perform better on the reduced data set which does not contain the schema information than on the original one. Our explanation for this is that the intersection graph contains part of the schema and thus produces a similar overlap for many of the instances.

5.6.5 Evaluation on Livejournal data

In a second setting, a larger dataset consisting of FOAF descriptions of people from the community website LiveJournal.com was used for experiments.

Class	#Inst.	Properties	#Inst.
Location	1344	*located*	3735
School	2794	*attend*	4118
ChatAccount	5	*holdsOnlineAccount*	3008
Person	22745	*knows*	116023
		hasImage	4554
Date	4	*dateOfBirth*	1567
#BlogPosts	5	*posted*	4872

Table 5.3: Number of known instances of classes and number of known instances of properties in the FOAF dataset.

Instance depth	1	2	3	D&D
#Instances		1567		1178
#Labels		32861		89
Average #nodes	35.26	550.15	4485.86	284.32
Max. #nodes	440	10779	28130	5748
Average #edges	34.26	934.04	12233.78	–
Max. #edges	439	17643	149234	–

Table 5.4: Statistics of the instance graphs in the SWRC dataset

Data set

Table 5.3 lists the classes, number of instances of each class (left column) and their properties and number of instances of each property (right column) used for experiments. Please note that *Date* and *#BlogPosts* are reduced to a small number of discrete states. E.g. the precise age has been replaced by one of four newly introduced age classes. In all our experiments the goal was to learn how people are to be classified into an age class.

For the evaluation, we removed all relations of type *dateOfBirth* from the dataset and tried to learn this relation afterwards, i.e. the goal was to predict for all 1567 people with available age information to which of the 4 age classes they should belong.

We report some statistics on the instance graphs with which we deal in our experiments in Table 5.4. Note that the graphs for instance depth 2 are already twice as big as the graphs in the D&D dataset by Dobson and Doig (2003) which is considered to contain large graphs and is used in the literature to evaluate mining techniques on large graphs.

Compared approaches

We have compared several configurations of the kernels proposed in Section 5.2. We have compared our kernels to the Weisfeiler-Lehman kernel (Shervashidze and Borgwardt, 2009) and the Gaertner kernel (Gärtner et al., 2003). Results of our experiments are reported in Table 5.6 (at the end of this chapter). For the Gaertner kernel it took two months of calculation time to obtain the results, while the other evaluation runs finished within few hours.

Discussion

Our results show that the walk kernel and the path kernel can outperform the basic edge kernel in terms of classification errors. However, in terms of F1 measures, performance deteriorates for structures with more than two edges. This effect is probably partly due to the way instance extraction was performed in our experiments: if there are walks with more than two edges in the graph, they are formed by cross-links in the instance graph, i.e. by nodes which are reached on different paths during the extraction process. As for the partial subtree kernel it turned out that low discount factors had to be chosen in order to obtain kernel values which can be expressed within the datatype `double` and thus be processed by SVM*light*. However, these low discount factors lead to kernel values which are relatively close to each other and are problematic with respect to finding an optimal solution in the SVM optimization problem: in our experiments we frequently encountered single classes for which the classifier would yield a 0% precision and recall. In this setting, the Weisfeiler-Lehman kernel performs comparably to our kernels and performance of our kernels deteriorates with increasing instance depth. We suspect that both is because age information can be derived mostly based on the node's direct neighborhoods, e.g. the people one directly knows. Traversing the social graph more deeply does presumably not provide additional relevant information.

5.7 Related Work

Starting with the work of Haussler (1999), research on kernel functions for structured data, i.e. for data that is expressed in a paradigm different from the standard vectorial representation, has become a major topic of investigation. In Section 5.1, we have already reviewed a set of kernel functions on graph structures and we have discussed the approach of Bloehdorn and Sure (2007) in the context of our evaluation. In this section, we complement this analysis by reviewing related endeavours in the area of kernel functions on semantic data structures like RDF.

As the first work in the direction of kernel functions for logic-based repre-

sentations, Gärtner et al. (2002) have proposed kernel functions on individuals represented as (closed) terms of the typed λ-calculus. While the kernel function can be flexibly tuned to different settings, the formalism used does not build on an established knowledge representation standard as, in our case, RDF. On the schema-level, Fanizzi and d'Amato (2006); Fanizzi et al. (2008a) propose a declarative kernel for semantic concept descriptions in the description logic (DL) $\mathcal{ALC}$. Structurally, these kernels are based on a representation of $\mathcal{ALC}$ concepts in normal form. The kernel is defined inductively by treating disjunctions (sums) and conjunctions (products) seperately. The similarity between atomic classes is measured in terms of the intersection of their extensions. The obvious restriction of this approach is that it only allows to compute kernel functions on concept descriptions but not on individuals. While Fanizzi and d'Amato (2006) suggest that the kernels on class descriptions could also be used for describing individuals by means of their most specific concepts, it does not provide any means to assess the characteristics of a given individual in terms of its object and data properties which – for the case of RDF – lie at the core of our approach. To overcome the dependency on DL languages, a different set of kernels, which can be applied directly to individuals, has been proposed by Fanizzi et al. (2008b). While these kernel functions come close to our approach, they rely on clean, DL-based formal ontologies which, in contrast to lightweight RDF-based data, only constitute a very small part of the "semantic" data sources published. Summing up, in contrast to the mentioned related work our approach starts with the raw RDF data and can handle noisy and sparse domains without making any assumption on the consistency of the data or manual specifications of semantic constructs.

Bicer et al. (2011) have defined a kernel function for RDF data which can also deal with these problems. Their kernel is based on the definition of ILP-clauses representing triple patterns. However, their approach requires an adaptation of the learning algorithm for choosing a relevant subset of the feature space. This adaptation has only been proposed for the case of relational learning (see Chapter 6), it is not trivial to adapt this selection step for the problem of entitiy classification that we discussed in this chapter.

5.8 Conclusion

With the advent of the Resource Description Framework (RDF) and its broad uptake, e.g. within the Linked Open Data (LOD) initiative, an increasing amount of graph-structured data has become available. In this paper, we have introduced a principle approach for exploiting RDF graph structures within established machine learning algorithms by designing suitable kernel functions. We have introduced two versatile families of kernel functions for RDF entities based

on intersection graphs and intersection trees and have shown that they have an intuitive, powerful interpretation while remaining computationally efficient. In an empirical evaluation, we demonstrated the flexibility of this approach and show that kernel functions within this family can compete with hand-crafted kernel functions and computationally more demanding approaches. Another lesson learned is that RDF graphs have strongly varying characteristics depending on the domain they are describing. Thus, our experiments suggest that it is not feasible to find the one single best kernel and parameter setting for RDF data in general. Every RDF graph as to be analyzed individually and according to this specific kernels can be recommended. However, as a general finding, the kernel functions based on intersection trees can be calculated more efficiently than those based on the analysis of the intersection graph. We also found that an instance depth of 2 leads to a good performance in most cases while the graphs can still be handled efficiently.

As an extension to this work, we plan further evaluations in order to obtain guidelines for the choice of the best kernel function for a specific application. Further substructures such as cycles may be considered. An interesting aspect is the integration of background knowledge in the form of manually defined kernels as proposed by Bloehdorn and Sure (2007).

Kernel configuration				Evaluation results			
Kernel	Instance Depth	Max. Size	Discount	Error	Precision	Recall	F1 measure
Bloehdorn and Sure (2007)				**0.0449**	0.9583	0.5813	**0.7237**
Shervashidze and Borgwardt (2009)	2	2		0.1096	0.0000	0.0000	0.0000
Gärtner et al. (2003)	2		0.5	0.0590	0.8740	0.5341	0.6625
Edges	1			0.1096	0	0	0
Edges	2			0.0576	0.9669	0.4692	0.6318
Edges	3			0.0576	0.9669	0.4692	0.6318
Paths	1	2	1	0.1096	0	0	0
Paths	2	2	1	0.0576	0.9669	0.4692	0.6318
Paths	3	2	1	**0.0478**	0.8259	0.675375	**0.7431**
Walks	1	2	1	0.1096	0	0	0
Walks	2	2	1	0.0576	0.9669	0.4692	0.6318
Walks	3	2	1	0.0815	0.7039	0.4983	0.5835
Full Subtree	1		1	0.0660	0.9792	0.3475	0.5129
Full Subtree	2		1	0.0548	0.9676	0.4943	0.6543
Full Subtree	3		1	**0.0463**	0.9699	0.5718	**0.7195**
Partial Subtree	1		1	0.1096	0	0	0
Partial Subtree	1		0.1	0.1096	0	0	0
Partial Subtree	1		0.01	0.1096	0	0	0
Partial Subtree	1		0.001	0.1096	0	0	0
Partial Subtree	2		1	0.1096	0	0	0
Partial Subtree	2		0.1	0.1096	0	0	0
Partial Subtree	2		0.01	0.1025	0.2500	0.4630	0.3247
Partial Subtree	2		0.001	0.1096	0	0	0
Partial Subtree	3		1	0.1096	0	0	0
Partial Subtree	3		0.1	0.1096	0	0	0
Partial Subtree	3		0.01	0.0810	0.2222	0.2222	0.2222
Partial Subtree	3		0.001	0.1096	0	0	0
Edges (no schema)	2			**0.0478**	0.8288	0.6828	**0.7488**
Walks (no schema)	2	2	1	**0.0478**	0.8288	0.6828	**0.7488**
Paths (no schema)	2	2	1	**0.0478**	0.8288	0.6828	**0.7488**
Full Subtree (no schema)	1		1	0.0660	0.9792	0.3475	0.5129
Full Subtree (no schema)	2		1	0.0548	0.9511	0.5156	0.6687
Full Subtree (no schema)	3		1	0.0492	0.9534	0.5526	0.6997

Table 5.5: Evaluation results for the SWRC dataset. Best configurations (with respect to accuracy and F1 measure) of compared graph kernels, intersection graph kernels, intersection tree kernels and for the dataset without schema information are marked in bold.

Kernel configuration				EvaluationResults			
Structure	inst. depth	maxSize	Discount	Error	Precision	Recall	F1 measure
Shervashidze and Borgwardt (2009)	2	2		**0.2215**	0.4174	0.3998	0.4084
Gärtner et al. (2003)[a]	2		0.5	0.3824	0.4841	0.5964	**0.5344**
Edges	2			0.2125	0.4246	0.3958	0.4096
Walks	2	2	0.5	0.1948	0.4774	0.3710	0.4175
Walks	2	2	1	0.1948	0.4774	0.3710	0.4175
Walks	2	2	2	0.1948	0.4774	0.3710	0.4175
Walks	2	3	0.5	0.1964	0.4738	0.3417	0.3971
Walks	2	3	1	0.1964	0.4738	0.3417	0.3971
Walks	2	3	2	0.1964	0.4738	0.3417	0.3971
Paths	2	2	0.5	0.1946	0.4785	0.3729	0.4192
Paths	2	2	1	**0.1946**	0.4785	0.3729	**0.4192**
Paths	2	2	2	0.1946	0.4785	0.3729	0.4192
Paths	2	3	0.5	0.1961	0.4741	0.3352	0.3927
Paths	2	3	1	0.1961	0.4741	0.3352	0.3927
Paths	2	3	2	0.1961	0.4741	0.3352	0.3927
Full Subtree	1		1	0.1902	0.4648	0.3538	**0.4018**
Full Subtree	2		1	0.2312	0.3986	0.3875	0.3930
Full Subtree	3		1	0.2532	0.3755	0.3967	0.3858
Partial Subtree	2		0.005	0.2039	0.3188	0.2364	0.2715
Partial Subtree	2		0.01	0.1866	0.4671	0.2534	0.3286
Partial Subtree	2		0.05	0.5874	0.2063	0.7422	0.3229
Partial Subtree	3		0.001	0.2240	0.1380	0.3125	0.1915
Partial Subtree	3		0.005	0.2009	0.3423	0.2342	0.2781
Partial Subtree	3		0.01	**0.1568**	0.3060	0.2535	0.2773

Table 5.6: Evaluation result for kernels based on intersection trees on the FOAF dataset - kernels are normalized. Best configurations (with respect to accuracy and F1 measure) of compared graph kernels, intersection graph kernels and intersection tree kernels information are marked in bold.

[a]The evaluation of of the Gaertner kernel took two months of calculation time, while the results for the other evaluation runs were obtained within hours

Chapter 6

Kernel Methods for RDF Link Prediction

In the previous chapter, we have presented methods for classifying RDF data. The goal in the classification problem was to assign entities from an RDF data set to one of a limited number of groups. The problems we considered where those of classifying people into age classes and to predict to which research group a person belonged.

However, these kind of problem statements are not quite in the spirit of RDF: The core idea behind RDF is to define entities and to link these entities among each other. In most cases, therefore, a lot more than the limited number of link targets which could be considered in the classification setting exist. Consider the affiliation problem: instead of only considering the members of the research groups at AIFB, we might be interested in classifying the members of the Semantic Web Research Community according to their affiliation. The high number of possible affiliations is prohibitive for applying classification methods to this problem. The stated problem could more efficiently be solved using *Link Prediction*.

Link Prediction is the problem of given a pair of entities (x, y) to decide whether this pair should belong to property p, i.e. whether a link should exist from x to y with label p. In the case of RDF data, the problem consists in predicting whether the triple (x, p, y) should exist in the dataset.

In the literature, mostly matrix-based methods have been used to solve this problem. Here, we focus on the question how kernel-based learning methods can be leveraged to predict links in RDF graph structures. While the problem setting of predicting links in graphs is not new (see e.g. Getoor and Diehl (2005)), the specific properties of RDF require a careful design of the employed kernel functions.

In the following, we introduce a general model for kernel-based mining of

relations for RDF graphs. We then instantiate the framework based on the kernel functions presented in the previous chapter. We evaluate and compare our method for link prediction to a state-of-the-art link prediction methods for RDF on two datasets.

The remainder of this chapter is organized as follows. In Section 6.1 we introduce the overall framework for kernel-based link prediction, discuss the typical problem of availability of positive training instances only, in Section 6.2 we describe how the kernels from Chapter 5 can be used for link prediction. In Section 6.4, we report on several experiments which demonstrate the flexibility of our approach and evaluate its performance in practical link prediction settings. We review related work in Section 6.5 and conclude with a discussion and an outlook on future extensions in Section 6.6.

6.1 Link Prediction using SVMs

One of the main challenges in link prediction for RDF data is the open-world assumption. Given an RDF knowledge base, only the expressed relations are known, whatever is not expressed in the knowledge base is not deemed false, but unknown. Thus, as RDF does not allow for negation, no negative data is available for the learning problem. We are thus confronted with a partially supervised learning problem.

Before presenting our method for link prediction, we will define some basic notations.

Definition 43 (Link prediction problem) *Given a RDF graph G and a property p, the problem of link prediction consists in finding a function f such that for any $(s, o), s \in domain(p), o \in range(p)$*

$$f((s, o)) = true \Leftrightarrow (s, p, o) \text{ should hold in } G.$$

We define the set of positive elements $\mathcal{P}$ as

$$\mathcal{P} = \{(s, o) | (s, p, o) \in G\}$$

Accordingly, the set of unknown elements $\mathcal{U}$ is defined as

$$\mathcal{U} = \{(s, o) | s \in domain(p), o \in range(p), (s, p, o) \notin G\}$$

6.1.1 Link Prediction with One-class SVMs

In order to overcome the problem that no negative training data is available, one-class classification models may be applied. This class of models is able to learn classifiers based on the training data from one class only.

We applied *One-class SVMs* (Schölkopf et al., 2001) to deal with this learning problem. This version of SVMs is given a set of input data belonging to one class and aims at finding a small region in the data which contains all the training examples. This is achieved by finding a hyperplane with maximum margin separating the training data from the origin in the feature space.

Formally, given a set of training data $x_1, \ldots, x_l \in \mathcal{X}$, the problem consists in finding a function f such that

$$f(x) = \text{sign}(\langle w, \phi(x) \rangle - \rho \geq 0$$

for most $x \in x_1, \ldots, x_l$. This is achieved by solving the optimization problem:

$$\min_{w \in F, \xi \in R^l, \rho \in R} \frac{1}{2}||w||^2 + \frac{1}{\theta l} \sum_i \xi_i - \rho$$

$$\text{subject to } \langle w, \phi(x_i)) \geq \rho - \xi_i, \xi_i \geq 0$$

The model learned by the One-class SVM can later be used for binary classification: Given an instance x, the classification is obtained by calculating $sign(f(x))$.

In the link prediction problem, all the instances from $\mathcal{P}$ may be used as training data, all tuples from $\mathcal{U}$ may be used for evaluation later on.

In some preliminary experiments we found that the approach using One-class SVM for training achieves very poor performance (about 50% accuracy and F-measure for a binary classification where there were as many instances from $\mathcal{P}$ as from $\mathcal{U}$ in the evaluation set). One possible explanation for this poor performance is that the origin in the feature space is not a suitable representation of the elements which are not part of the relation.

To improve on the method proposed so far, one option thus is to adapt the classifier training such that it includes a better approximation of where samples of the negative class should lie in the data space.

6.1.2 Link Prediction with Two-class SVMs

To overcome the problem of learning appropriate class borders for the relation, negative training data is needed. As RDF has open-world semantics, not stating that $(s, o) \in p$ does not mean that $(s, o) \notin p$, but that the membership status of (s, o) with respect to p is unknown.

To get negative training data nonetheless, we make the assumption that a tuple (s, o) is more likely to truly belong to p if (s, p, o) is part of the data graph than if it is not stated explicitely. Using this assumption, we propose to use a subset of the triples from $\mathcal{U}$ as negative training set, i.e. we randomly choose a number of instances that are assumed to be negative for the training process.

Binary classifier training works best with stratified samples, i.e. if positive and negative data are balanced in the data set. Therefore, as many positive as

negative instances are used for the training. As relations are usually sparse in RDF data, this number seems to give a good balance between the number of assumptions made and the validity of the training data.

6.2 Kernel Functions for Links in RDF

Each kernel machine requires a suitable kernel function which compares instances among each other.

Definition 44 (Link Kernel) *Given two potential instances* (s_1, o_1) *and* (s_2, o_2) *of a property p, the Link Kernel is defined as:*

$$\kappa((s_1, o_1), (s_2, o_2)) = f(\kappa_s(s_1, s_2), \kappa_o(o_1, o_2))$$

where κ_s, κ_o *are valid kernel functions. This means that the similarity of two instances of a relation is determined by the similarity of the instances' subjects and the instances' objects.*

The advantage of the link kernel is the low complexity of calculating the kernel matrix. Given a property p with d elements in the domain and r elements in the range, there are dr instances in the relational learning problem. While a classical kernel operating directly on the relation instances would require $(rd)^2$ kernel computations, i.e. the complexity of calculating the whole kernel matrix would be in $O(r^2d^2comp(\kappa))$ where $comp(\kappa)$ is the complexity of calculating the kernel function, using our approach only d^2+r^2 kernel function computations are needed. The kernel matrix can thus be computed in $O(r^2d^2+(d^2+r^2)comp(\kappa))$.

Obviously, f has to be chosen in a way such that it is guaranteed that the resulting function is a kernel function. The space of valid kernel functions is known to be closed under sum, product and multiplication with positive scalars, thus we can define the Sum Link Kernel and the Product Link Kernel as valid kernel functions.

Definition 45 (Sum Link Kernel, Product Link Kernel) *Given two kernel functions* κ_s *and* $kernel_o$ *for RDF entities and* $\alpha, \beta > 0$*, the* Sum Link Kernel *is defined as:*

$$f(\kappa_s(s_1, s_2), \kappa_o(o_1, o_2) = \alpha\kappa_s(s_1, s_2) + \beta\kappa_o(s_1, s_2)$$

Accordingly, the Product Link Kernel *is defined as:*

$$f(\kappa_s(s_1, s_2), \kappa_o(o_1, o_2) = \alpha\kappa_s(s_1, s_2) * \kappa_o(s_1, s_2)$$

The importance of subject and object for the kernel value may be changed using α *and* β*.*

Any kernel function which is applicable to entities in an RDF graph is a candidate for usage in the kernel. We have presented a set of such kernel functions in Chapter 5. Any of the kernel functions based on the intersection graph resp. the intersection tree may thus be used as subject or object kernel in the Sum Link Kernel and the Product Link Kernel.

6.3 Learning with Statistical Unit Node Sets

In the following, we will compare our approach to an alternative approach for Link Prediction, namely the approach for learning with Statistical Unit Node Sets (SUNS) proposed in Huang et al. (2010). Their model for link prediction is based on a multivariate prediction problem, i.e. on a supervised problem where the value of a set of variables is predicted at once.

Based on the available data, a data matrix is constructed. The rows of this matrix are defined by the elements of the population under consideration, e.g. the people working at a specific institute. The elements of the population are called statistical units. The columns are defined based on the triples in which a statistical unit participates. E.g., if a statistical unit is defined for a person `John`, then any of the triples in which `John` occurs defines a feature which takes the value 1 for John and any other statistical unit participating in a triple of this pattern, 0 for any other statistical unit. An example feature is `(A,knows,Jane)` which is 1 for any person knowing `Jane`. Additional attributes are defined for patterns where one of the constants is replaced by a variable and for the conjunction of a general triple pattern with a specific one. An example of one of the latter patterns is `(A,knows,B) and (B,hasIncome,High)`. The obtained data matrix is pruned: any columns having ones in less than ϵ percent of the data matrix or in more than $100 - \epsilon$ percent of the rows. The remaining features are those whose values are predicted in the model.

The prediction is then obtained after matrix completion. Huang et al. (2010) have proposed different methods which may be used for the matrix completion. In our evaluations we have used singular value decomposition (SVD). The approach for completing the matrix M is to decompose it into $M = SUV$ where U is a diagonal matrix. The completed matrix is obtained by setting all but d of the entries of U to 0 and by then calculating $M' = SU'V$.

6.4 Evaluation

We have evaluated our approach on two data sets and have compared it to the approaches based on statistical unit sets. Our implementation is based on libSVM (Chang and Lin, 2011) and uses the ν-SVM for training classifiers with different parameter settings. For the Statistical Learning approach we use an

implementation provided by the authors.

In this section, we will first present the evaluation methodology used, then we will present each of the evaluation scenarios and the results we obtained from them.

6.4.1 Evaluation Methodology

The open-world assumption does not only pose problems for choosing an appropriate learning algorithm, but also in the evaluation of the learned model: again, no negative data is available for testing the classifier.

Recall that for the training of the classifier we assumed that links that are explicitely stated are more likely to be true than links that are not explicitely stated. Our evaluation of link prediction methods is also based on this assumption: Instead of using the models to obtain an explicit classification, we use them to rank the data. The method is deemed to perform well if the positive examples in the hold-out set are ranked higher than examples which are not made explicit.

To obtain robust performance measures, we use 5-fold cross-validation in all evaluation settings. These folds are obtained by splitting the positive data into 5 parts of equal size. Each fold is then completed with as many negative examples as positive examples are contained in the fold which are also used in the training phase.

In total, 5 classifiers are trained, each using 4 of the folds for training. The learned model is then evaluated using the 5th fold and all the negative elements which are not part of any of the folds. The splitting of the data is visualized in Figure 6.1. Note that this approach is generally applicable because relations in RDF datasets are usually quite sparse, i.e. there are much more negative examples than positive ones.

As evaluation measures, we use two ranking measures established in the Information Retrieval literature: NDCG (Järvelin and Kekäläinen, 2000) and bpref (Buckley and Voorhees, 2004). The general idea of ranking evaluation is that a perfect ranking is obtained when all positive elements are ranked higher than any negative element. The idea behind the NDCG is to punish negative elements which are ranked higher than some positive ones by the position at which the negative element occurs in the ranking. The bpref punishes negative elements that are ranked higher than positive ones through the information how many negative elements are ranked higher than a positive one. Figure 6.2 shows an example ranking for instances of the `foaf:knows` relation in our example.

Definition 46 (NDCG) *Given a ranking of instances, the Discounted Cumu-*

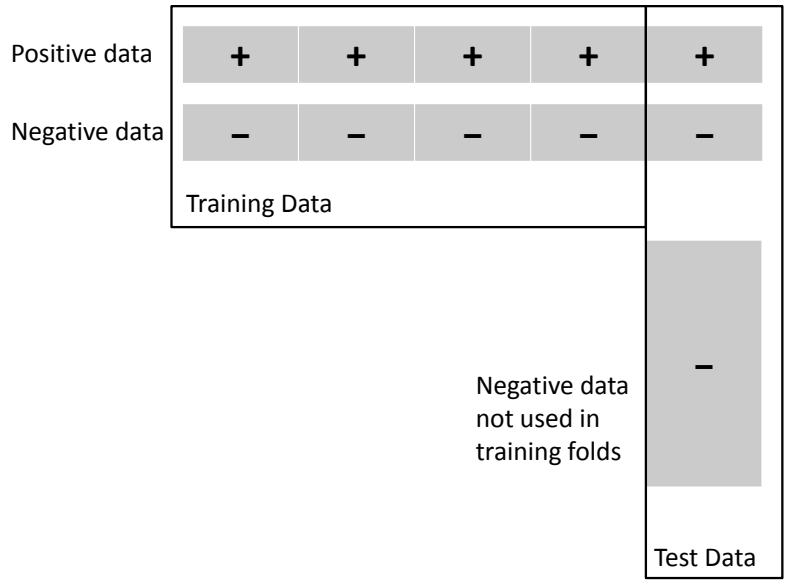

Figure 6.1: Illustration of the splitting of available data in training and test folds

lative Gain (DCG) at position p is defined as

$$DCG_p = rel_1 + \sum_{i=2}^{p} \frac{rel_i}{\log_2 i}$$

where

$$rel_i = \begin{cases} 1 & \textit{a positive example is found at position } i \textit{ in the ranking} \\ 0 & \textit{otherwise} \end{cases}$$

The Normalized Discounted Cumulative Gain (NDCG) at position p is obtained as

$$NDCG_p = \frac{DCG_p}{IDCG_p}$$

where $IDCG_p$ describes the Ideal Discounted Cumulative Gain and stands for the DCG_p obtained for the ideal ranking.

Calculating the NDCG for the maximal position in the ranking, i.e. for the whole dataset yields a measure which has its highest value when all the positive instances are ranked higher than the negative ones. It also takes into account the positions at which the relevant data was found.

The second evaluation measure we are using, Bpref, was specifically designed for evaluating rankings with incomplete information (Buckley and Voorhees, 2004):

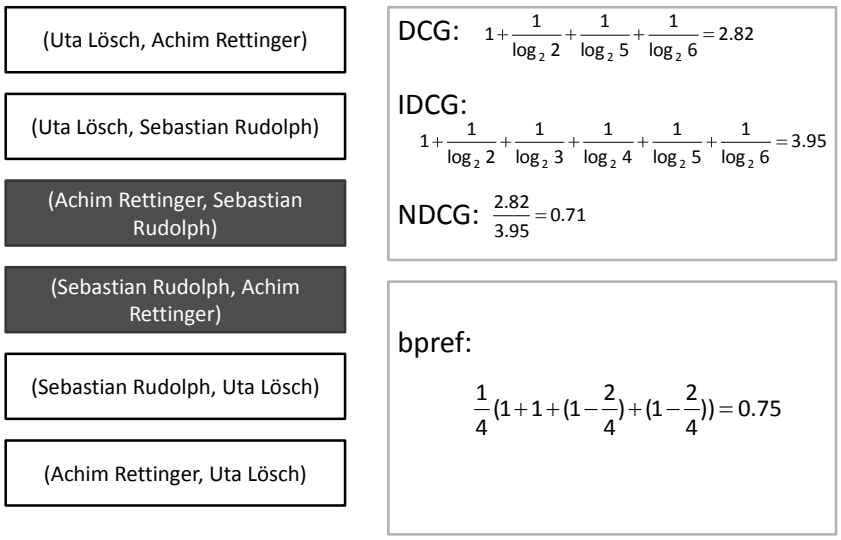

Figure 6.2: Example ranking of `foaf:knows` instances and NDCG and bpref measures. The ranking is to be read from top to bottom, black boxes indicate unknown instances, white boxes known instances.

Definition 47 (bpref) *For a ranking with R relevant elements where r is a relevant element and n is a negative element within the first R elements in the ranking,*

$$bpref = \frac{1}{R}\sum_{r} 1 - \frac{|n \text{ ranked higher than } r|}{R}$$

Basically, the bpref counts how many negative resp. unjudged elements were ranked higher than each of the positive elements in the ranking. The optimal value of the bpref is 1, in the worst case it is 0.

6.4.2 Learning affiliations in the SWRC dataset

On this relatively small data set, the evaluation goal is an assessment of the influence of different parameters on the learning result.

The evaluation uses the data from the SWRC ontology (Sure et al., 2005) and the metadata which is available in the Semantic Portal of the institute AIFB. The same dataset has been used for evaluations in Chapter 5. The ontology models key concepts within research communities, among them people, publications, projects and research topics. The evaluation data consists of 2,547 entities of which 1,058 can be derived to belong to the person class. 178 of these

Kernel configuration				Evaluation Results	
Kernel	Inst.depth	α/β	ν/Dimensions	NDCG	bpref
SUNS-SVD			10	0.7455(± 0.0384)	0.3034(± 0.0569)
SUNS-SVD			20	**0.8022**(± 0.0606)	**0.3844**(± 0.0661)
SUNS-SVD			30	0.7293(± 0.0836)	0.3364(± 0.0953)
SUNS-SVD			50	0.6524(± 0.0683)	0.2558(± 0.0479)
SUNS-SVD			100	0.5609(± 0.0834)	0.1958(± 0.0570)
Common Subtrees	2	1	0.1	0.4241(± 0.0761)	0.0283(± 0.0004)
Common Subtrees	2	1	0.2	0.5108(± 0.2274)	0.2060(± 0.3482)
Common Subtrees	2	1	0.3	0.7409(± 0.2714)	0.5478(± 0.4192)
Common Subtrees	2	1	0.4	0.9467(± 0.0252)	0.8357(± 0.0639)
Common Subtrees	2	1	0.5	0.9340(± 0.0209)	0.8438(± 0.0743)
Common Subtrees	2	1	0.6	0.9390(± 0.0218)	0.8360(± 0.0648)
Common Subtrees	2	1	0.7	0.9446(± 0.0212)	0.8284(± 0.0584)
Common Subtrees	2	1	0.8	0.9416(± 0.0487)	0.8119(± 0.1255)
Common Subtrees	2	1	0.9	**0.9572**(± 0.0308)	**0.8644**(± 0.0865)

Table 6.1: Evaluation Results for Link Prediction on SWRC dataset with 95% confidence intervals

persons are affiliated with one of the research groups at AIFB, 78 of them being currently employed. Additionally, there are 1232 instances of type publication, 146 instances of type research topic and 146 instances of type project. The entities are connected by a total of 15,883 relations among them. Additionally, there are 8,705 datatype properties, i.e. properties linking an entity to a literal.

In the evaluation we are learning the `affiliation` relation. Note that this setting has been used in other evaluations (see Chapter 5 and Bloehdorn and Sure (2007)), however there the setting was to learn a classifier which would decide given a person which research group this person should belong to. Our goal here is slightly different. Given a tuple consisting of a person and a research group, the goal consists in deciding whether this tuple is part of the `affiliation` relation and hence whether the person is affiliated with the given group.

Our results (see Table 6.1) show that the kernel-based approach can achieve significantly higher NDCG and bpref values than the approaches based on matrix completion. When analysing the results of the matrix-based approaches we found that in some cases no probability value could be assigned to a possible link. We assume that this is because the respective rows of the data matrix were pruned during the training process. This also explains the relatively low bpref of the matrix-based approaches: any positive element of the test set for which no classification can be found is ranked lower than any of the negative instances in the test set for which a prediction is found.

The results of the Subtree Kernel indicate that the results of the classification improve with growing trade-off factor ν. This means that our models improve when the model complexity is increased which means that even the complexest models that were learned with our approach generalize well and do not suffer from overfitting the data. It therefore seems that in the proposed feature space

Kernel configuration				Evaluation Results	
Kernel	Inst. depth	α	β	NDCG	bpref
Common Subtrees	2	0.5	1	0.9316(± 0.0182)	0.8363(± 0.0654)
Common Subtrees	2	1	1	0.9340(± 0.0209)	0.8438(± 0.0743)
Common Subtrees	2	2	1	0.9548(± 0.0314)	0.8538(± 0.0857)
Common Subtrees	2	3	1	0.9271(± 0.0607)	0.7831(± 0.1570)
Common Subtrees	2	5	1	0.8387(± 0.1135)	0.6313(± 0.2863)

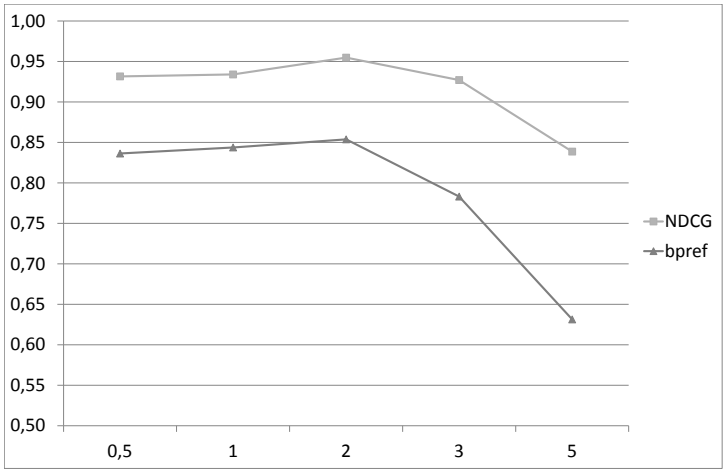

Figure 6.3: Influence of the quantity α/β on the classification result, 95% confidence intervals in parentheses

the elements of the relation are well separated from but close to the negative instances. This is also supported by the high variance of the models learned based on a ν-value of 0.2 or 0.3.

We have analysed the influence of the relation between parameters α and β on the result (see Figure 6.3). In the case presented here, optimal results were obtained for $\alpha/\beta = 2$. However, in the given application scenario, there are only few possible objects of the relation. It is therefore intuitive that the similarity of the subjects should have higher impact on the classification.

6.4.3 Learning friendships in Livejournal data

In the second evaluation setting we studied features of a larger data set and a relation which is quite different from the `affiliation` relation in the SWRC dataset. In a dataset which has been extracted from the social network site LiveJournal[1] we attempt to learn the `foaf:knows` relation. A similar dataset has been used for evaluations in Chapter 5, however, here we only use a subset

[1] `http://www.livejournal.com`

Class	Instances
Person	638
Age	4
Location	200
School	747
OnlineAccount	5
BlogActivity	5

Table 6.2: Statistics on the LiveJournal dataset

System configuration					Evaluation Results	
Kernel	Inst.depth	α	β	ν/Dimensions	NDCG	bpref
SUNS				20	**0.7871**(± 0.0101)	**0.2889**(± 0.0175)
SUNS				50	0.7410(± 0.0071)	0.2191(± 0.0075)
SUNS				100	0.6847(± 0.0022)	0.1446(± 0.0043)
Common Subtrees	2	1	1	0.1	0.9397(± 0.0100)	0.8085(± 0.0271)
Common Subtrees	2	1	1	0.2	0.9746(± 0.0019)	0.9149(± 0.0073)
Common Subtrees	2	1	1	0.3	0.9736(± 0.0029)	0.9135(± 0.0063)
Common Subtrees	2	1	1	0.4	0.9712(± 0.0040)	0.9072(± 0.0079)
Common Subtrees	2	1	1	0.5	0.9886(± 0.0137)	**0.9680**(± 0.0394)
Common Subtrees	2	1	1	0.6	0.9794(± 0.0071)	0.9066(± 0.0079)
Common Subtrees	2	1	1	0.7	0.9828(± 0.0070)	0.9186(± 0.0143)
Common Subtrees	2	1	1	0.8	0.9895(± 0.0059)	0.9422(± 0.0273)
Common Subtrees	2	1	1	0.9	**0.9916**(± 0.0079)	0.9580(± 0.0362)

Table 6.3: Evaluation Results for Link Prediction on Livejournal dataset with 95% confidence intervals

of the dataset described there. The main difference between the two scenarios besides the size of the dataset is the number of elements in the range of the property: while in the SWRC dataset only 5 elements were in the range of the learned property, here there are as many objects in the domain as in the range of the property.

Our dataset consists of descriptions of 638 people and their friendship relations. Overall, there are 8069 instances of the `foaf:knows` relation. Additionally, the dataset contains information on the people's location, schools they attended, other online accounts they hold etc. The data set was cleaned and some of the relation values, like those for the `age` relation were aggregated. Overall statistics of the dataset are listed in Table 6.2. In total, there are 3040 nodes and 15907 relations in the dataset. The average node degree is 5.2326.

The results obtained on this dataset confirm our findings for the Semantic Web Research Community (SWRC) dataset: our approach leads to significantly better results than the matrix approaches. The relatively low bpref of the matrix completion approaches indicate that no classification was obtained for some of the positive elements of the test set. Again, our results improve with a growing trade-off parameter ν in the support vector machine, thus indicating the good

Kernel configuration				Evaluation Results	
Kernel	Inst. depth	α	β	NDCG	bpref
Common Subtrees	2	0.5	1	0.9724(± 0.0038)	0.9056(± 0.0054)
Common Subtrees	2	1	1	**0.9886**(± 0.0137)	**0.9680**(± 0.0394)
Common Subtrees	2	2	1	0.9694(± 0.0186)	0.8948(± 0.0245)
Common Subtrees	2	3	1	0.9741(± 0.0066)	0.9006(± 0.0085)
Common Subtrees	2	5	1	0.9744(± 0.0055)	0.8945(± 0.0078)

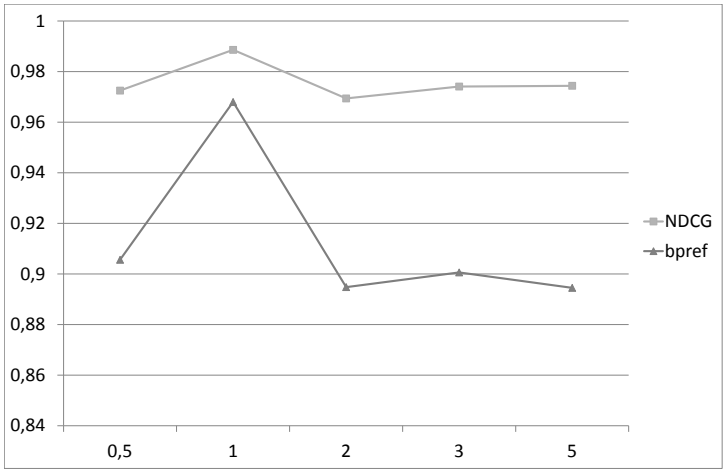

Figure 6.4: Influence of the quantity α/β on the classification result, 95% confidence intervals in parentheses

separation of positive and negative examples in the training set.

We have also analysed the influence of the relation between α and β on the classification result (see Figure 6.4). On this dataset the best results are achieved when subject and object similarity get equal importance. The `foaf:knows` relation has as many potential subjects as objects. It is therefore natural that the object should have more influence on the classification than in the case of the `affiliation` relation which only has a very limited number of objects.

6.5 Related Work

Link Prediction is a problem which has previously been studied not only in the context of RDF data (see Getoor and Diehl (2005) for an overview). An important application of Link Prediction is the recommendation of links in social networks (Liben-Nowell and Kleinberg, 2003; Yin et al., 2010).

Applications of Link Prediction to Semantic Web data have only recently been studied. Besides the SUNS approach (Huang et al., 2010) to which we

have compared our approach, Rettinger et al. (2009) have also proposed the application of Statistical Relational Learning methods to this problem. Their approach allows for the integration of hard constraints made available through an ontology into the Infinite Hidden Relational Model (Xu et al., 2006).

Kernel-based methods for Link Prediction in the context of Semantic Web data have been proposed by Bicer et al. (2011). They define so-called clause kernels which each use a single triple pattern as feature. These clause kernels are then combined into a global kernel function. The optimal weights for the clause kernels are learned through a combination of the classical SVM training algorithm with genetic algorithms. This means that an adaptation of the learning algorithm is required in their approach, while our approach can be used with any kernel machine and is thus also applicable to other scenarios where learning from links is desired.

6.6 Conclusion

In the context of semantic data which is described by an underlying graph structure, link prediction, i.e. learning whether a link of a specific type should exist between two entities of interest, is one of the most important learning problems.

In this chapter, we have proposed an approach for adapting entity classification approaches to the link prediction problem. We have instantiated our general method using the entity kernel functions proposed in Chapter 5.

The proposed link prediction method has been evaluated in two evaluation settings and compared to matrix-based multivariate prediction methods. We could show that our approach outperforms these approaches.

Chapter 7

Knowledge Base Updates

7.1 Introduction and Motivation

One aspect of Ontology Refinement is the need for maintaining the information in a way such that the domain model is consistent with the underlying domain. This requires new information to be added when it becomes available, to detect the implications of new information on the current data and to find those statements that should be changed due to the newly added information. Furthermore, new information may allow for the inference of additional information that is not expressed explicitely, which may also be added to the knowledge base.

As of now, little support for these problems is available. However, by providing suitable support in form of a partial automation of the update process would facilitate updating a knowledge base in a way that keeps it consistent with the domain and within itself and to find interdependencies between the data more easily. Additionally, the update process could be made more easy to handle (requiring less expertise) and more efficient.

To address this problem, we have developed a mechanism for ontology and knowledge base updates, which enables the automatic handling of frequently recurring updates. An example for this kind of updates is a research domain where the employees of a research group, their projects, publications and supervisor relationships are modelled. A recurring update in this setting is a person finishing her PhD thesis. The changes required to turn a PhD student into a Post-Doc in the knowledge base are similar for every person. Thus, they may be taken care of in the same way. Other examples are people joining or leaving the research group.

The method we propose is in general applicable in an automatic or a semi-automatic way. In very clearly structured domains, it may be sufficient to trigger the change mechanism with some parameters and the update can be performed

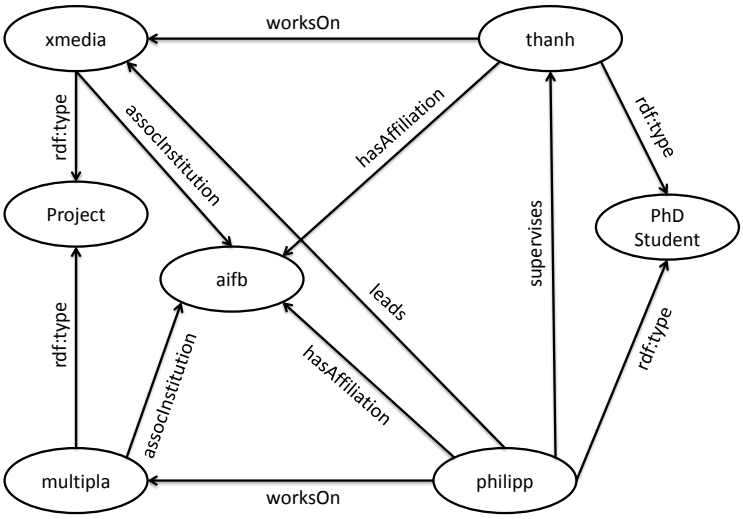

Figure 7.1: Sample from the research community ontology

automatically. In other cases, in which the changes are more ambiguous the change mechanism may be used in an interactive way which allows the user to determine the single changes which are applied to the ontology.

Throughout this chapter, we will motivate our design choices and implementation details based on a running example. Consider an ontology which models a research community, projects and supervisor relationships. A sample from this ontology is given in Figure 7.1. We consider the case of somebody - in the example `philipp` - leaving the institution. This means that the type of event is `LeaveInstitution`. In the case that somebody leaves the institution the person should not be affiliated with this institution anymore and she should not work on any projects there any longer. As explained above, we aim at defining a system which performs these changes automatically once it is known that the person has left the institution.

The remainder of this chapter is structured as follows: in Section 7.2 we introduce the notion of Ontology Updates and distinguish it from related domains such as Ontology Revision, Section 7.3 explains our design choices when working out the overall and more detailed aspects of an ontology update language. In Section 7.4 we briefly sketch the architecture of a system where ontology update specifications are employed. Section 7.5 provides the formal specification of our proposed Ontology Update Language. In Section 7.6 the benefits of our approach will be illustrated by means of a small example. In Section 7.7 we will present an interactive extension of our approach. Section 7.9 describes the

details of our reference implementation of an according ontology update system , before we conclude and provide directions for future research in Section 7.10.

7.2 Ontology Update Support

The comprehensive modeling of complex domains requires domain knowledge and ontology modelling expertise. As few people have both, ontologies are rarely developed by a single person starting from scratch. Rather collaborative design and development of ontologies and continuous refinement are the usual scenarios for which elaborate methodologies and appropriate tool support are crucial (see for example (Tudorache et al., 2008; Tempich et al., 2007)).

In this spirit, *ontology change* is a fundamental issue to be addressed. Ontology Changes may occur for various reasons (see Chapter 4). One reason for changing the ontology is a change in the underlying domain. We called these changes *Ontology Updates*.

Ontology Updates may be identified by answering the question

Is the ontology changed due to an according change in the described domain?

Note that there are many cases where the answer to that question would be no: an ontology change might be the consequence of the discovery of modeling errors (*ontology repair*) or the acquisition of new additional domain knowledge (*ontology refinement*). Obviously, the latter type of ontology changes reflects a change in the way the modeler conceives or formalizes the domain of interest. As an example, consider the case that new findings in genetics might imply that the taxonomy of living beings has to be corrected in order to properly reflect the current knowledge of the real situation. A lot of research has been devoted to this kind of ontology change (employing techniques from belief revision, knowledge acquisition, ontology learning and ontology evolution to name just a few, see Chapter 4 for an overview).

As opposed to those, we will be concerned with the task that we refer to as *(temporal) ontology update*: changes to the ontology might become necessary as the underlying domain changes over time. As time passes, the state of affairs in the domain like a person's employer or her academic title may change. While this kind of changes will mostly concern assertional knowledge, they may also concern the schema. As an example consider EU membership: it is expected that additional countries become member of the EU, thereby changing the terminological definition of the class representing the European Union. The term ontology update is thus used in the sense of literally keeping an ontology up-to-date.

In many cases it will be possible to come up with *change patterns* which describe typical ways in which a domain may evolve. For example, a domain

individual recorded as underage may turn into an adult, while the opposite is impossible. This example illustrates that change patterns are domain specific and depend on the intended semantics of the involved domain entities: on the abstract structural level, both `Child` and `Adult` are just classes and hence on par with each other.

While most of these patterns will probably deal with the update of assertional knowledge, they may also occur on the schema level (as an example consider a country becoming member of the EU). While here we focus on change patterns that reflect temporal and domain-specific changes, we are aware that such patterns may be identified beyond this use case (our approach may thus be applicable in other change scenarios, too).

The observation that such change patterns exist leads to the idea to formally specify typical ways in which an ontology may be updated over time. As an example, in a biological domain, an individual might cease to be member of the class `Caterpillar` and become member of the class `Butterfly` instead. These update specifications may concern schema knowledge as well as fact knowledge.

On a more general level, update specifications allow to encode process knowledge and associate it with the ontology, such that it can be used for updates. Thus, by adding change patterns to the ontology, a dynamic domain model may be obtained: while the ontology describes the state of the domain, the change patterns describe how the model may evolve in the future.

The specifications can be seen as operational descriptions how to update an ontology as a consequence to information entered into the system. However, in contrast to common Ontology Evolution approaches (Stojanovic, 2004), we propose to base those updates on domain specific knowledge about temporal changes. Using the above mentioned information, a natural reaction to asserting that an individual `willie` is now a class member of `Butterfly` would be to also retract the information that `willie` is an instance of `Caterpillar` (Carle, 1969).

This way, an ontology change requested by a person responsible for ontology maintenance can be supplemented by additional changes. This allows to prevent modeling flaws that might occur due to only partially entered information. Generally, an ontology update specification allows constraining ontology changes to clearly defined, foreseen cases in a domain-specific way.

Then, less knowledge of the concrete formalization of the domain is needed in order to make updates, as it is sufficient to trigger an update specification for a specific kind of change without knowing all the implications the change may have on other parts of the ontology. Thus, frequent maintenance or update tasks can be transferred from knowledge engineers (roughly: the "ontology administrator") to knowledge workers (possibly formally less skilled users in charge of monitoring changes in the domain of interest and transferring them into the ontology) while minimizing the risk of introducing errors. In our example, it would be sufficient

that somebody enters the information that `philipp` is not affiliated with the `aifb` anymore into the system, all additional changes that are necessary will be determined by the system. It is not necessary that the person entering the new knowledge in the ontology is familiar with the formalisation of the domain.

Like the actual ontology, the according update specification has to be created by a knowledge engineer who is also in charge of ontology refinement activities as well as addressing unforeseen changes not anticipated by the update specification that might become necessary.

7.3 Design Choices

In this section, we review the major choices and questions to be addressed when designing a framework for ontology update management.

7.3.1 Ontology-inherent Temporal Knowledge vs. External Specification

One approach to capture temporal changes in a domain is to use logic formalisms that allow for their description inside ontological specifications. There is a plethora of formalisms and approaches such as temporal logics (see (Lutz et al., 2008)) or situation calculi (Levesque et al., 1998) that provide a logic-inherent way of describing temporal and dynamic phenomena in the domain. Clearly, these approaches have advantages whenever the intended use of the ontology includes reasoning over domain changes (maybe even planning). However, besides the more complicated formalisms, a usual drawback of such formalisms is the high reasoning complexity.

Using such a formalism would mean to include temporal axioms such that a person has a specific type until another type is assigned. In our example an axiom could define that a person is a PhDStudent until she has published a PhD thesis.

Note that our goal is much more moderate: from the above described, it becomes clear that we aim at designing an operational formalism that – given a change request – deterministically comes up with an updated ontology in a timely manner. Moreover, we would like to stick to the usual approach that an ontology encodes a static description of the domain, hence every state of the actual ontology should be considered a kind of "snapshot".

Therefore, we adopt an approach keeping the actual ontology and the according update specification distinct, which also enables the use of off-the-shelf reasoners for dealing with the ontology part.

This means that the formalism describing the updates is not part of the ontology itself but outside of it. The ontology stays as is and is associated with an

external specification which contains descriptions of the types and effects of updates. In our example this external description contains a specifications of kind `LeaveInstitution` which specifies that when somebody leaves an institution she also stops working on projects there.

An additional advantage of specifying changes externally is that standard tools for ontology management may be used instead of special implementations for non-standard formalisms.

7.3.2 Unguided Belief Revision vs. Guided Update

Although most of the work done in the area of belief revision has dealt with scenarios of ontology refinement (see (Konstantinidis et al., 2008) for an RDFS-based formal framework and (Qi and Yang, 2008) for a survey), some proposals have been made to also address the update scenario (Katsuno and Mendelzon, 1992).

Notwithstanding, belief revision approaches try to resolve inconsistencies that were introduced by an update. However, many changes in the ontology will not lead to an inconsistent state, thus no additional changes are performed.

Therefore, the applicability of belief revision is restricted to formalisms expressive enough to cause inconsistencies. While this is certainly the case for OWL, causing "meaningful" inconsistencies in RDF(S) is virtually impossible. More precisely: in RDF(S) inconsistencies can only be provoked by so-called XML-clashes, which is more a datatype-related unintentional peculiarity than a design feature. To a certain extent, this fallback can be mitigated by adding additional constraints beyond RDF(S) on top of an RDF(S) knowledge base

As another downside of belief revision, note that the strategies to restore consistency do not take domain specifics into account. To illustrate that, consider the following example: let a knowledge base contain the disjointness of the classes `Adult` and `Child` and the fact that `Peter` belongs to the class `Child`. If we now add that `Peter` is also an instance of the class `Adult`, the knowledge base becomes inconsistent and (if configured appropriately) a belief revision approach would retract `Child(Peter)`, as newly added facts override those already present. While this is the desired behavior, re-adding `Child(Peter)` to the new knowledge base would lead to the deletion of `Adult(Peter)` irrespective of the actual irreversibility of this development in the described domain. Clearly, a more appropriate "reality-aware" reaction of an update mechanism would be to reject the second change request.

As opposed to belief revision, our approach aims at preventing inconsistent ontology states that might arise from incomplete change requests. To this end, change requests are completed by further ontology changes based on specified knowledge about how a domain may develop. This way, consistency can be preserved; as a worst case, the change request can be denied. E.g., in our example,

the request for removing `philipp hasAffiliation aifb` does not lead to an inconsistency. However, the state of the ontology if only this statement is removed is an unintended one as people should not work on projects at institutions they are not affiliated with.

Note however, that those two approaches are not mutually exclusive but could be combined: in case of a change request not matching any of the anticipated update patterns, applying a belief revision "fallback solution" may be preferable to simply rejecting the request.

7.3.3 Syntactic vs. Semantic Preconditions

In most cases, the best way to react to a change request will depend on the current state of the ontology. Hence it is crucial to provide the opportunity to formulate respective preconditions for triggering changes. There are essentially two distinct kinds of checks that can be done against an ontology: *semantic* and *syntactic* ones (this distinction has been proposed by Vrandečić (2010), however what we call syntactic here is called *structural* there).

If some changes should be made depending on the validity of some statement in the ontology, we have to employ reasoning in order to decide whether the statement is logically entailed by the given information. As a special case of this, one could diagnose whether an intended change would turn the ontology inconsistent and reject the requested change on these grounds. Semantic checks provide the more thorough way of testing the knowledge contained in an ontology, however the reasoning to be employed may be expensive with respect to memory and runtime.

The alternative would be to just syntactically determine whether certain axioms are literally contained in a knowledge base. This would be less expensive than the semantic approach. Yet usually, there are many possible ways to syntactically express one piece of semantic information making a naïve syntactic "pattern matching" approach problematic at best. One way to mitigate that problem while still avoiding to engage in heavy-weight reasoning would be to syntactically normalize the ontology and the change request. That is, the ontology is transformed into a semantically equivalent, but syntactically more constrained form, facilitating to identify and manipulate pieces of semantic information by purely syntactic analyses (of course, this does not make reasoning superfluous, but it can at least be avoided in some cases).

Since both kinds of preconditions are useful under different circumstances, we argue that an ontology update formalism should offer both options leaving to the knowledge engineer to decide which one should be used in a specific case. However, in our actual proposal, we refrain from taking a purely syntactic approach, but rather the structural level of the RDF graph that is used by SPARQL.

7.3.4 Transparency

We think that an update mechanism should be transparent for classical ontology modelling tools. Thus, the ontologies which have an update formalization attached can be processed by ontology tools such as reasoners which are not built for automatically processing updates.

There are two advantages of this design decision: first, no special modelling features are needed, but the domain model can be defined using classical ontology languages. The update specification is external to the ontology itself and can be processed by tools that are dealing with this aspect. However, simple direct updates remain possible in cases where the ontology is processed by tools other than an update processing tool. Thus, no special tools are needed for tasks not related to the update specification and execution. Secondly, existing ontologies may be amended with update specifications without changing the environment in which they are used.

Another aspect with respect to which we propose transparency is with respect to the formalism used for stating updates. It is thus possible to issue the same change requests as when no update specification is available. If the user or the external tool changing the ontology is not aware of the update mechanism being in place, it may completely ignore its output. In the case of RDF, this means that updates should be stated using SPARQL update. In case no update mechanism is implemented the change request `philipp hasAffiliation aifb` only removes this exact triple. If however the update mechanism is available, the triples stating that `philipp` leads resp. works on projects and supervises students at AIFB are also removed.

7.3.5 Change Feedback

It has to be expected that the changes mediated by an ontology update specification might not be directly obvious for the knowledge worker. However, it is clearly crucial to ensure that the system's behavior is as comprehensible as possible to the knowledge worker. For this reason, feedback about the automated reactions to a change request should be an essential part in any practically employable ontology update framework.

In order to provide informative feedback, the ontology engineer has to provide template-like explanation snippets commenting on the nature of the change patterns contained in the update specification and the changes triggered by them. At runtime, those templates instantiated with the actually changed ontology elements can be presented to the knowledge worker in order to explain what actually happened to the ontology. In our running example the knowledge worker applying the change should obtain information on which statements were removed additionally, e.g. a message like this:

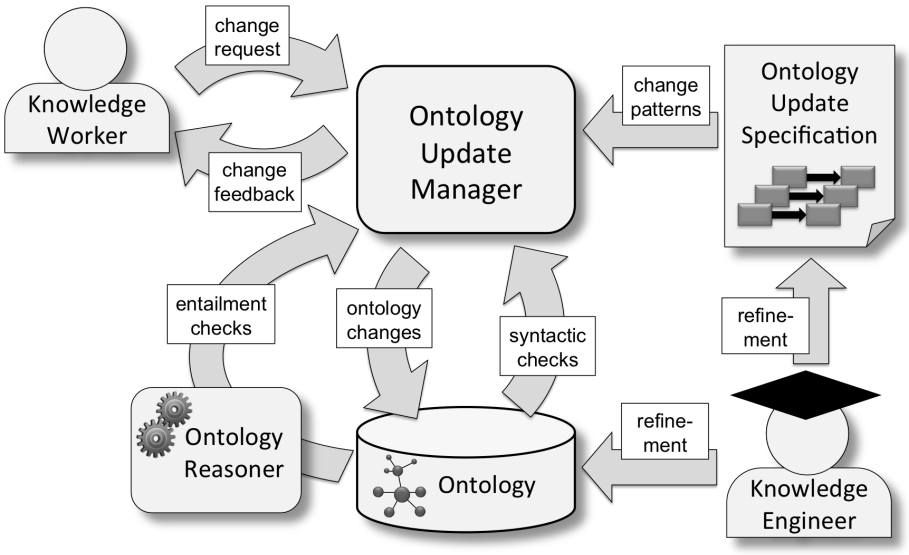

Figure 7.2: Ontology Update Architecture

```
   Philipp does not work at AIFB anymore.
Thus, he does not lead xmedia anymore.
He has stopped working on multipla.
Thanh is not supervised by Philipp anymore.
```

As the knowledge worker may not be well acquainted with the concrete formalization of the domain, it may be advisable not to output the changed statements directly but to also give an explanation as to why a triple is changed. For example, in the case of someone leaving an institution, a useful output could be that the person is no longer working on any project at the institution before listing the changes that occur due to that condition.

7.4 System Architecture

In this section, we propose an abstract architecture for an ontology update framework taking into account the design choices made in the previous section. A concrete instantiation of this architecture is described in the subsequent sections. The suggested architecture is sketched in Fig. 7.2.

Therein, the usual unguided interaction mode of committing changes directly to the ontology is complemented by an additional update management component as editing interface for the knowledge worker. Still, the knowledge engineer will be able to directly change and refine both the ontology and the ontology update specification.

The typical work flow of an ontology update step is carried out as follows: Initially, the knowledge worker issues a change request by providing a piece of

knowledge to be added to or deleted from the ontology.

A change request will only be acted upon if the accompanying Ontology Update Specification accounts for it. The Ontology Update Manager will scan the Update Specification for an update rule whose applicability conditions are satisfied by the uttered change request and the ontology. Those applicability conditions might contain syntactic as well as semantic checks. If there are several applicable update rules, only one of them is applied (in our implementation the one that was first registered with the Ontology Update Specification).

If an applicable update rule has been determined, the change request can be acted on accordingly by denying or accepting it but possibly also by carrying out more ontology changes than explicitly requested. In the present version this happens automatically without additional explicit approval of these additional changes.

Finally, a feedback message describing the activated change pattern and containing the actually performed changes is generated and sent back to the knowledge worker.

In our implementation the change request is denied if no applicable update specification is found. It is however logged such that the ontology engineer can take care of it later and also refine the Ontology Update Specifications if needed. In case there are several matching update rules, the first matching rule is triggered. The ontology engineer thus has to take care in which order the update rules are specified. It may however be envisioned to give the knowledge worker the choice between several update specifications if more than one specification matches the given request.

The proposed framework is inspired by database triggers as e.g. described in the SQL standard (Date and Darwen, 2008). Database triggers are stored procedures that are activated by changes that are submitted to the database. They may be defined for insertions, updates and deletions of instance data in the database. Our approach provides the same kind of functionality for instance data in ontologies while additionally allowing for defining update specifications for schema changes, which is usually not possible with database triggers.

In general, database triggers can be specified to be executed before or after committing the submitted change. While we do not offer this possibility, we offer the possibility to activate changes based on entailments after the change has been applied, thereby allowing changes to be performed based on what will be changed in the ontology. To evaluate these conditions the ontology obtained by directly applying the requested change (without the additional changes) is evaluated.

Effects of the update specification might depend on when the change is performed (before or after the other actions specified in the update specification). We therefore decided to give maximum flexibility to the ontology engineer by

```
CREATE CHANGEHANDLER <name>
FOR <changerequest>
AS
  [ IF <precondition>
  THEN ] <actions>
<changerequest>  ::== add    [unique] (<SPARQL>)
                  |   delete [unique] (<SPARQL>)
<precondition>   ::== contains(<SPARQL>)
                  | entails(<SPARQL>)
                  | entailsChanged(<SPARQL>)
                  | (<precondition>)
                  | <precondition> and <precondition>
                  | <precondition> or <precondition>
<actions>        ::== [<action>]|<action><actions>
<action>         ::== <SPARQL update>
                  | for( <precondition> ) <actions> end;
                  | feedback(<text>)
                  | applyRequest
<SPARQL>         ::== where clause of a SPARQL query
<SPARQL update>  ::== a modify action (in SPARQL Update)
<text>           ::== string (may contain SPARQL variables)
```

Figure 7.3: Ontology Update Language syntax specification in BNF.

having him specify when the submitted change should be executed. To make this possible, we decided not to execute the submitted change at all. Instead it has to be specified in the update specification when the change is to be applied.

7.5 A Language Proposal

In this section, we instantiate our previous general considerations by providing an ontology update framework based on RDF(S) and SPARQL, as well as SPARQL Update. This framework consists of the syntax of the Ontology Update Language (OUL, specified in Fig. 7.3) together with the precise description how ontology change requests are to be handled by the ontology management component (see Algorithm 6).

Every update rule (also called changehandler) has an identifier. It carries a change request pattern, expressing for which change request it can be applied and some preconditions that define whether a change request can be handled by this rule depending on the current ontology state. If several matching changehandlers exist, the first one occurring in the update specification will be applied. We are aware that we thereby deviate from pure declarativity. However, for a first proposal, this kind of implicit priority declaration seems both intuitive and

computationally feasible.

A change request pattern is defined by the `WHERE`-clause of a SPARQL select query that is to be evaluated on the change request. That means that the statements submitted for a change are interpreted as an RDF graph and it is checked whether the change request pattern executed as SPARQL query yields any (or, if the `unique` option is set: exactly one) result on this change graph (lines 4 – 6 in Algorithm 6). The result of this query are bindings of all variables that are present in the `WHERE`-clause. Those bound variables can be reused later in the preconditions and actions part. Using SPARQL for describing changes leads to the transparency that we have discussed to be useful. If no Ontology Update Specification is available the changes can be handled by a regular SPARQL endpoint.

If a match is found, the precondition of the changehandler is evaluated. This determines whether the changehandler is applicable or whether another matching one has to be found. There are three basic types of preconditions: a syntactic check (that simply verifies whether certain triples are contained in the ontology) is performed via `contains`. Semantic entailment checks may be performed on the ontology in its current, unaltered state (`entails`) or on the "hypothetical" ontology that would result from carrying out the changes as requested (`entailsChanged`). As explained above, it is desirable to provide syntactic and semantic checks on the ontology, as syntactic checks are less expensive but also less accurate than semantic checks. Syntactically, basic preconditions are also the `WHERE`-part of a SPARQL query.

Basic preconditions can be combined by `and` and `or`. As it is reasonable to allow for (yet unbound) variables to occur in several basic preconditions, the `and`- resp. `or`-operators are realized as join and union on the result sets of the basic preconditions. In the end, a precondition is considered to be successful (line 8) if its result set contains at least one entry. Before evaluation of the precondition, all variables that occurred before in the change request pattern are substituted by their binding (line 7, in the case of several existing bindings, the first one is chosen).

If the precondition is evaluated successfully, the actions specified in the changehandler's body are applied to the ontology. It is up to the knowledge engineer what should be done with change requests for which no matching changehandler is found. This can be done by specifying a changehandler which matches any change request.

As for the evaluation of preconditions, all variables occurring in the action part of the changehandler that were bound before (i.e. that were present in either the change request or in the precondition) are replaced by their binding before the action part is applied (line 16). If several possible bindings were found for the change request or the precondition, the first binding is chosen.

Elementary actions that can be carried out are knowledge base changes (expressed as SPARQL Update operations), the `applyRequest` action (carrying out the originally uttered change request), and feedback messages. Moreover, elementary actions can be nested into loops which iterate over the result set of a precondition. While executing the action part of a changehandler, the activated ontology changes are not directly applied but first assembled in a list (line 17) and applied thereafter. This way all the loop preconditions are evaluated against the original ontology (or, in the case of `entailsChanged`, against the ontology that has been altered in the initially proposed way), thereby preserving a declarative flavor. The application of the changes is done in an atomic manner after all necessary changes have been determined.

As it would not be easy to ensure termination or avoid high computational cost if the *actions* part of a change request was allowed to trigger other change requests, no other changehandlers are triggered during the execution of a changehandler. While this ensures termination, it makes the ontology engineer responsible for "manually" handling all additional changes that might become necessary due to the changes during the execution of the changehandler.

As a preliminary solution, the association of changehandlers with an ontology works similar to the association of DTDs with an XML document (Bray et al., 2008). They can either be defined inline in the document specifying the data or they can be defined in external files. In either case, the changehandler is defined in a comment (such that the RDF file can also be parsed by ontology management systems that do not support OUL). All comments that have an extra '-' at the beginning are parsed as changehandler definitions. This begin of the comment may be followed by a file name enclosed in quotation marks, which defines an external changehandler, or by the definition of a changehandler enclosed in square brackets, defining the changehandler inline.

7.6 Examples

In this section, we provide a set of examples aimed at both advocating the potential usefulness of our proposed update framework and demonstrating the concrete work flow.

7.6.1 Running Example

We start with the knowledge base from Fig. 7.4 which is equivalent to the RDF graph presented in Figure 7.1. Additionally, an update specification for the ontology is available which contains the definition of a change handler which is used for processing events of type `LeaveInstitution` and `authorsPhD` (see Fig. 7.5). The first changehandler therein deals with the case that somebody leaves his/her current affiliation. In that case, the deletion of the affiliation

```
philipp     rdf:type            PhDStudent .
philipp     hasAffiliation      aifb .
philipp     leads               xmedia .
philipp     worksOn             multipla .
philipp     supervises          thanh .
thanh       rdf:type            PhDStudent .
thanh       hasAffiliation      aifb .
thanh       worksOn             xmedia .
xmedia      rdf:type            Project .
xmedia      assocInstitution    aifb .
multipla    rdf:type            Project .
multipla    assocInstitution    aifb .
```

Figure 7.4: Example knowledge base.

information has to trigger further changes: the person will not continue to lead projects at the institution he/she leaves nor to supervise persons. Now suppose the following change request, indicating that Philipp is leaving the AIFB institute, is entered into the system: `delete data {philipp hasAffiliation aifb .}`

The system will now check whether this change request matches the first changehandler's change pattern `del { ?x hasAffiliation ?y }`. This is the case, as the corresponding SPARQL query yields a result which binds `philipp` to `?x` and `aifb` to `?y`.

The considered changehandler is now executed as it does not contain further preconditions for activation. So, the specified actions will be carried out: `applyRequest` means that the initial change request is granted and added to the list of updates to be executed. After that the following message is displayed: `philipp is no longer affiliated to aifb.`

Next, we consider the graph pattern in the first loop. Note that it contains variables that have already been bound by the change pattern matching. Before evaluating the loop action, those variables are substituted by their bindings, in our case resulting in the following rewritten loop action:

```
for(contains(philipp ?wol ?z . ?z rdf:type Project .
             ?z  assocInstitution aifb .
             FILTER(?wol=worksOn || ?wol=leads)))
   delete data {philipp ?wol ?z};
   feedback("philipp does not lead/work on project ?z anymore"); end;
```

Now, the conditional part of the rewritten loop action is matched against the database, yielding the following two variable bindings: `?wol`↦`leads`, `?z`↦`xmedia`

```
CREATE CHANGEHANDLER leavesInstitution
   FOR del { ?x  hasAffiliation  ?y }
AS applyRequest;
   feedback("?x is no longer affiliated to ?y");
   for(contains(?x ?wol ?z . ?z rdf:type Project .
                ?z  assocInstitution ?y .
                FILTER(?wol=worksOn || ?wol=leads)))
      delete data {?x ?wol ?z};
      feedback("Thus, ?x does not lead/work on project ?z anymore."); end;
   for(contains(?x  supervises ?z .  ?z  hasAffiliation ?y))
      delete data {?x supervises ?z};
      feedback("Thus, ?x does not supervise ?z anymore"); end;

CREATE CHANGEHANDLER authorsPhD
   FOR add { ?x  swrc:authorOf  ?y }
AS IF entailschanged( ?y  rdf:type swrc:PhDThesis . )
   THEN applyRequest;
        delete data { ?x  rdf:type  swrc:PhDStudent};
        feedback("Change accepted. ?x authored a PhDThesis,
                  so he is no PhD student anymore.");
```

Figure 7.5: Example ontology update specification.

and `?wol`↦`worksOn`, `?z`↦`multipla`. Next, for each of these two bindings, the subsequent actions are executed: therefore, the triples `philipp leads xmedia.` and `philipp worksOn multipla.` are scheduled for deletion and the following two messages are prompted to the user:

```
Thus, philipp does not lead/work on project xmedia anymore.
Thus, philipp does not lead/work on project multipla anymore.
```

In analogy to that, by executing the second loop of the activated changehandler, `philipp supervises thanh.` is scheduled for deletion and the message

```
Thus, philipp does not supervise thanh anymore.
```

is prompted to the user. Finally, all the scheduled changes are carried out.

In addition to this complete example we will present some standard situations or decisions which might occur in an ontology update setting and how they can be realized by means of the Ontology Update Language as presented in this paper.

7.6.2 Restricting the Size of the Change

Clearly, it is not always desirable to permit change requests of arbitrary size. In principle, an entire ontology could be added in one step, which would be a situation hard to handle with update rules. One solution to this is to restrict the size of the change a priori. As an extreme case of this, only one RDF triple per change might be allowed. While this constraint can be imposed by external means, our formalism is flexible enough to handle it. In order to allow only add changes consisting of one triple, the following changehandler would have to be inserted at the beginning of an ontology update specification:

```
CREATE CHANGEHANDLER tooMuchForOneBite
   FOR add ( { ?a ?b ?c . ?d ?e ?f .
       !(sameTERM(?a,?d) && sameTERM(?b,?e) && sameTERM(?c,?f))} )
   AS feedback("Request denied. Only one triple per change!");
```

In words, the change request pattern checks whether there are two distinct triples contained in the change request. If so, the changehandler is activated without doing any changes (thereby effectively rejecting the request). Note that the implemented selection strategy also prevents any subsequent changehandler in the update specification from being activated.

7.6.3 Handling of Change Requests

Of course, change requests might occur which do not activate any of the specified changehandlers. In the presented implementation, the request will be tacitly denied in this case. It is however possible to create changehandlers that match any `add` (resp. `delete`) request. Placed at the bottom of an ontology update specification, those can be used to provide feedback whenever no other changehandler was activated:

```
CREATE CHANGEHANDLER noMatchRestrictive FOR add ( { ?a ?b ?c . } )
AS feedback("Request denied. No matching change rule found!");
```

This is just an explication of the default restrictive strategy: every unforeseen request will be denied. In the same way, it is of course possible to realize a permissive strategy by stating

```
CREATE CHANGEHANDLER noMatchPermissive FOR add ( { ?a ?b ?c . } )
AS applyrequest;
   feedback("Request accepted. No matching change rule found.");
```

instead. This way, all requests not matching any of the preceding changehandlers will be complied with.

7.7 Interactive Ontology Updates

We have adapted and extended the Ontology Update Language presented in the previous sections to a more interactive setting (see Kayser (2010) for a complete presentation of the approach). The idea is that the same change can be issued due to different reasons which imply different changes. For example, the change request

```
delete data {philipp  rdf:type  PhDStudent.}
```

may be due to Philipp having earned his PhD, or due to Philipp having quit his PhD studies. The motivation of the change request determines which additional triples should be updated. If Philipp has finished his PhD, his type should be changed from `PhDStudent` to `PostDoc`, a title for his PhD thesis should be entered. In case Philipp did not finish his PhD, he will most likely change employers, and his status of being a `PhDStudent` should be deleted, although it depends on Philipp's concrete next steps what his new type should be.

In interactive OUL, the goal is to adapt changehandlers such that they can deal with different *change causes*. The idea is to issue a set of questions which the user has to answer in order to determine all the changes needed.

In the interactive version of OUL, the scenarios of deletion and addition of data are differentiated. When a triple is added to the knowledge base, which contains a resource that has not been used before, the user is questioned for its type. Depending on the type of resource which is added, a template is chosen which is used to obtain additional information about the resource. For example, in the scenario of modelling a research institute, additional information which might be needed is the project(s) on which the person is working, a phone number, the office where he has his workplace etc.

In the case of deletion of data, three cases are distinguished:

- Should only the statement itself be deleted?
- Should the subject of the statement be removed from the knowledge base?
- Should the object of the statement be removed from the knowledge base?

Note that this distinction between different motivations for deleting from the ontology are domain-independent, the specification of how the change should be dealt with are domain-dependent. An example for the first type of change is that a person switches projects, in which case the relation between him and his former project should be replaced by a relation between him and his new project. Another reason for deleting the project affiliation could be that the person has become the leader of the project. We therefore propose to define a set of questions for determining the effective changes that are needed which

depends on the property of the deleted statements. Note that the three reasons given above are independent from the domain of interest. However, the changes triggered based on these reasons are domain-dependent.

In the second and the third case the resource itself has to be deleted and thus every statement which has this resource as its subject or object has to be deleted and may have to be replaced. An example for this kind of change is a project which has finished and which is to be deleted from the knowledge base. In this case, no one should be working on the project anymore. However, people who have previously worked on the project should be associated with another project.

For these cases, a deletion handler for each class has to be defined[1]. This deletion handler handles all kinds of relations a resource of a specific type can have and also defines whether relations of a specific type have to be replaced or can plainly be deleted.

Thus, in the interactive setting, the process of processing a change request has to be changed. Additionally, the transparency of the mechanism has to be given up, as the user has to enter a dialogue with the system in order to determine the whole set of changes which have to be executed.

In the overall architecture of the system only the step of choosing applicable update specifications has to be changed. In case information is added to the knowledge base, a check is required as to whether the subject and the object are already part of the knowledge base. If one or both of them are new resources, the user is asked for the class the new resource belongs to, the templates for the specific class(es) are chosen and executed.

In case of deleting data from the knowledge base, the user is asked whether the deletion concerns the subject, the object or only the relation itself. Depending on the answer a deletion handler for the class of the deleted resource respectively for a triple with the given property is chosen.

The deletion handlers themselves are built in the same way they were built in the case withouth interactive elements. The grammar has to be extended to allow for asking questions and processing answers to them.

7.8 Related Work

The contribution of this chapter consists in the definition of an update mechanism which enables a partial automation of frequently recurring updates within the ontology. To the best of our knowledge no other proposals for a similar mechanism exist. However, there is some research which is concerned with ontology

[1] Inheritance may be exploited in the change handlers, i.e. a deletion handler may call the deletion handlers of the superclasses of the class for which it is specified. However, this is not implemented in the current system

change.

However, Papavassiliou et al. (2009) have proposed to analyse change logs of RDF/S knowledge bases in order to generate high-level descriptions of the changes that are made to the ontology. The idea is to abstract from the addition and deletion of triples to more abstract descriptions such as the generalization of the domain of a relation. They propose a set of high-level changes into which each set of atomic changes can uniquely be divided. Their approach is domain-independent as the proposed high-level changes are not dependent on the concrete formalization at hand. OUL could be used for encoding the high-level changes and would thus allow for directly carrying out a high-level change.

Ontology Evolution deals with the problems arising from changes in the schema of the ontology and their propagation to dependent artefacts. Stojanovic (2004) has defined *evolution strategies* which are dealing with inconsistencies arising from changes to the ontology and which help to automatically resolve these problems. Again, the proposed solutions are on the level of constructs of the ontology language and do not incorporate domain-specific information.

The field of *Ontology Versioning* is dealing with the problems arising from the evolution of an ontology and due to changing and updating and ontology with respect to model specific kind of data. One main aspect is to grasp semantic differences between two versions of an ontology and to maintain information as to whether a concept or relation has the same meaning in two versions of the ontology (Klein and Fensel, 2001; Klein et al., 2002; Völkel and Groza, 2006). The challenge of finding this information consists in the possibility to express the same information in several ways and in the fact that not all information is represented explicitely in the ontology but may be inferred using reasoning tools. The overlap between the problems in ontology versioning and the change management we propose is that we are also interested in whether a change would induce specific changes to the semantics of concepts in the ontology.

Ontology Revision Qi and Yang (2008) deals with automatically detecting inconsistencies in an ontology and in automatic means for resolving them. Standard strategies consist in finding a minimal subset of the axioms from the ontology which is still inconsistent. Removing one of the axioms leads to a consistent ontology. The problem of ontology revision in the context of guided changes is that it is not clear a priori which of the axioms are removed during the revision process.

7.9 Implementation

We provide an implementation of our architecture and our language proposal at `http://people.aifb.kit.edu/uhe/OUL/`. The implementation uses Jena as underlying framework for ontology management. This framework was chosen,

as Jena provides an implementation of SPARQL Update (Schenk et al., 2008).

SPARQL Update is an extension of the ontology query language SPARQL (Prud'hommeaux and Seaborne, 15 Jan. 2008). While SPARQL's purpose is to find data in a RDF graph, SPARQL Update provides functionality for updating and managing RDF graphs using a SPARQL-like syntax (see Section 2.3 for an introduction to SPARQL Update).

Our implementation provides a wrapper for Jena's SPARQL Update endpoint, which implements the ontology update management as we proposed it. SPARQL update requests can be submitted as in the original implementation, but instead of directly executing them, the graph of changes that will have to be applied is constructed and a suitable change handler is searched for as explained in Section 7.5 and the respective actions are performed. By default, if no change handler is found, the ontology remains unchanged. This approach makes the update management as transparent as possible for the user. The only difference with respect to using the original implementation is using another endpoint and getting all the described advantages. This allows to open SPARQL endpoints for writing access more liberally.

7.10 Conclusion

In this chapter, we have addressed the task of updating an ontology due to changes in the described domain. We have argued for a formalism that allows for specifying the domain-dependent ways in which a specific ontology may evolve over time. We thoroughly discussed the crucial design decisions to be made for an ontology update framework that would automatically align change requests with change patterns and thereby allow to delegate simple ontology maintenance tasks to users not necessarily possessing the expertise of a knowledge engineer. We presented and implemented a proposal for such a framework.

An elaborate ontology update mechanism as presented here allows ontologies to be updated in a more predictable and quality preserving way. Administrators of ontology based systems may choose to allow a wider audience to edit their ontologies in a controlled manner, thus extending the collaborative aspect of ontology maintenance.

Algorithm 6: Processing of Change Request

Input: ontology $\mathcal{O}$ consisting of axioms (RDF triples – Note that in RDFS every axiom is represented by exactly one triple), ontology update specification $\mathcal{US}$ treated as list of changehandlers, change request $op(Ax)$ where $op \in \{\texttt{add}, \texttt{del}\}$ and Ax is a set of axioms resp. triples.

Data: *candidate* changehandler that is checked for applicability
toExecute container to store the activated changehandler
updateList list of SPARQL Updates to be carried out, initially empty

Result: Updated ontology $\mathcal{O}$

```
1  //find an appropriate changehandler
2  while toExecute.isEmpty and not US.endOfDocument do
3      candidate ← US.nextChangeHandler
4      matches ← SPARQLmatch(candidate.changerequest, op(Ax))
5      if not matches.isEmpty then
6          if matches.count == 1 or not candidate.changerequest.unique
           then
7              instPrecondition ←
               Substitute(candidate.precondition, matches.first)
8              if not evaluate(instPrecondition, O).isEmpty then
9                  toExecute.add(candidate)
10             end
11         end
12     end
13 end
14 //execute actions, if applicable
15 if not toExecute.isEmpty then
16     todo ← Substitute(toExecute.first.actions, matches.first)
17     cumulateActions(O, todo, updateList)
18     foreach update ∈ updateList do
19         apply update to O
20     end
21 end
22 return O
```

Part IV

Conclusion

Chapter 8

Conclusion

In this chapter, we will first summarize the results of this thesis including how the research questions defined in the beginning were adressed and which solutions were presented to solve the identified problems. Afterwards, we present an outlook based on questions which were raised but not answered in this thesis. These open questions may give rise to further research in the future.

8.1 Summary

This thesis has addressed the Ontology Change process. First, an overview of existing approaches for the support of different changes and different phases of the change process was given. We have identified two shortcomings of existing approaches: the lack of support for Ontology Updates, i.e. for changes which originate in a change in the domain, and the lack of methods for Ontology Mining which address the specific properties of ontology languages with low expressivity, especially RDF.

Based on these shortcomings of existing work, two research questions have been defined and addressed:

1. How can uncertain facts be induced from lightweight ontologies in a domain-independent way?

2. How can knowledge bases be updated automatically, such that new information can automatically be incorporated into the knowledge base without making it inconsistent or incoherent?

The contribution of the thesis consists in the proposal of a family of kernel functions for RDF data, which has proven to give results which perform comparably to state-of-the-art kernels for entities in semantic data. In contrast to semantic kernels which were previously defined, the proposed kernel functions

rely on the analysis of the graph structure represented by the ontology, thus being independent of the semantic interpretation of the knowledge representation. Additionally, this allows for an off-the-shelf application of the proposed kernel methods to new domains without adaptation of the kernel functions to the new domain.

Besides the application of the kernel functions for classifying entities in RDF datasets, the proposed kernel functions have also been applied to the link prediction problem on which they were compared to statistical relational learning approaches. Our evaluation shows that our kernel functions can outperform the statisical relational learning approaches.

With respect to the support of Ontology Updates, we have identified change patterns, i.e. types of changes, as a useful starting point for supporting the Ontology Update process. The observation that changes may be grouped into classes which describe the same kind of change and which lead to similar changes in the ontology, motivates the definition of a framework for exploiting these frequent patterns in the changes. So-called changehandlers specify for a class of changes how they should be processed and which additional changes they should trigger. The changehandler describes the changes necessary to adapt the ontology to a specific kind of change in the domain. A framework of how these changehandlers may be incorporated has been proposed. While the framework itself is independent of the precise knowledge representation, it was implemented for the case of RDF in the scope of this thesis. A language for specifying change patterns based on RDF and SPARQL Update has been proposed. An interactive extension of the proposed framework allows for the handling of classes of changes which may occur for different reasons and require different actions depending on the reason for which they occur.

8.2 Future Work

While the work presented here presented here are first steps towards a solution of the problems that were identified at the beginning of the thesis, there is still a lot of space for improvement. In the following, we discuss some lines of research which may be interesting to pursue in order to improve the systems and approaches presented here.

8.2.1 Kernel Functions for RDF data

The kernel function presented here are based on the analysis of the graph structures underlying the RDF data representation.

Combination of simple kernels. In our experiments, we have examined the performance of kernel functions based on the analysis of a single kind of structure. However, the combination of different kernels using kernel modifiers

such as weighted sums may improve the results of the obtained classification. An interesting aspect therefore would be the combination of our kernels with manually designed kernel functions such as the ones proposed in Bloehdorn and Sure (2007), which allow for the encoding of additional background knowledge and assumptions into the kernel.

Choice of appropriate kernel. Our experiments show that within the family of kernel functions we propose no single best kernel function exists and that the existing approaches show a very unstable performance when applied to different data sets. So far, the reasons for these differences in performance are only understood to a very limited extent. For the future development of machine learning methods for the Semantic Web, it would be beneficial to analyse the properties of RDF data sets and to identify those which influence the performance of learning algorithms on these datasets.

Automatic ontology refinement. The kernel methods presented in Chapters 5 and 6 may be used to train classifiers which are then used for completing an ontology. The thus derived links may be integrated in the ontology using the ontology update framework defined in Chapter 7. The classifier would thus become the issuer of change requests sent to the update framework.

Concept Drift detection deals with detecting changes in the distribution of the values of the predicted variable in supervised learning. The classifiers for link prediction presented in Chapter 6 may be used to detect the change of usage in a certain type of link (by comparing the distribution of the predicted values for all instances over time) and can thus trigger a request for manual inspection of the ontology, as an engineer might want to adapt the definition of the relation to its actual usage or the relation has been used in an error-prone way and manual adaptation is also possible. Such a kind of monitoring is especially interesting in the case where updates are performed automatically, e.g. in the update framework presented in Chapter 7.

8.2.2 Ontology Updates

Being aware that the framework, language and implementation presented in Chapter 7 constitutes just a first step towards a suitable trigger functionality for semantic technologies, we identify several directions for future research:

Extending the implementation to OWL. Currently, our implementation works with RDF(S) and SPARQL. Extending it to OWL would require to extend SPARQL accordingly, and to allow Algorithm 6 to use multiple-triple axioms as they often occur in OWL DL knowledge bases.

Combination with belief revision. Although we have argued that the rationale of belief revision does not fit well with our purpose, there are certainly cases where a combination of both is beneficial. Belief revision could be used as a fall-back strategy if a change request would lead to an inconsistent ontology and

is not tackled by any of the update specification's change patterns. Instead of simply rejecting the change, belief revision techniques could be more appropriate. In general, belief revision and other coping strategies could be incorporated into the proposed formalism in a plugin-manner as additional actions next to adding and deleting axioms.

Higher order constructs. Our proposal of an ontology update has a rather operational flavor. While this arguably facilitates the employment and allows for an efficient and straightforward implementation, a more declarative way of describing the possible domain changes would be more in the spirit of the current ontology languages. Moreover a specification in OWL would abide by the rationale to reuse formalisms (just as the XML syntax is also used for XML Schema).

Hence it seems sensible to introduce a more abstract description layer for complex changes, preferably in OWL. The underlying model for such a framework could be inspired by the usual ways of describing discrete dynamic systems such as finite automata or petri nets.

A simple example would be to relate the two classes Child and Adult with each other with a property allowing the transformation of instances of the one class to an instance of the other, e.g `Child disjointTransformationTo Adult`. Note that in OWL2 such a property is legal due to punning.

Learning Change Patterns. Clearly, the success of the proposed framework depends on the quality of the update specification. While in certain domains the development of such a specification might be straight forward (possibly because there are already informal documents describing the standard processes and work flows) there might be scenarios where this is not the case. Under those circumstances, frequent hange patterns could be extracted from ontology change logs by some machine learning techniques. Those findings could then be presented to the knowledge engineer as suggestions for ontology update rules to be incorporated into the specification.

List of Figures

List of Tables

Bibliography

Mohsen Afsharchi, Behrouz H. Far, and Jörg Denzinger. Ontology-guided Learning to Improve Communication between Groups of Agents. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multi-agent Systems (AAMAS 2006)*, pages 923–930, New York, NY, USA, 2006. ACM.

Franz Baader, Bernhard Ganter, Baris Sertkaya, and Ulrike Sattler. Completing Description Logic Knowledge Bases Using Formal Concept Analysis. In Manuela M. Veloso, editor, *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI 2007)*, pages 230–235, Hyderabad, India, January 2007.

Franz Baader et al., editor. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2007.

Sidney C. Bailin and Walt Truszkowski. Ontology Negotiation between Intelligent Information Agents. *The Knowledge Engineering Review*, 17(01):7–19, 2002.

Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, May 2001.

Veli Bicer, Thanh Tran, and Anna Gossen. Relational Kernel Machines for Learning from Graph-Structured RDF Data. In Grigoris Antoniou, Marko Grobelnik, Elena Paslaru Bontas Simperl, Bijan Parsia, Dimitris Plexousakis, Pieter De Leenheer, and Jeff Pan, editors, *Proceedings of the 8th Extended Semantic Web Conference (ESWC 2011)*, volume 6643 of *Lecture Notes in Computer Science*, pages 47–62, Heraklion, Greece, 2011. Springer.

Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked Data - The Story So Far. *International Journal on Semantic Web and Information Systems*, 5 (3):1–22, 2009a.

Christian Bizer, Jens Lehmann, Georgi Kobilarov, Sören Auer, Christian Becker, Richard Cyganiak, and Sebastian Hellmann. DBpedia – A Crystallization

Point for the Web of Data. *Journal of Web Semantics: Science, Services and Agents on the World Wide Web*, 7(3):154–165, 2009b.

Stephan Bloehdorn and York Sure. Kernel Methods for Mining Instance Data in Ontologies. In Karl Aberer, Key-Sun Choi, Natasha Noy, Dean Allemang, Kyung-Il Lee, Lyndon J B Nixon, Jennifer Golbeck, Peter Mika, Diana Maynard, Guus Schreiber, and Philippe Cudré-Mauroux, editors, *Proceedings of the 6th International Semantic Web Conference and the 2nd Asian Semantic Web Conference (ISWC 2007 + ASWC 2007)*, volume 4825 of *Lecture Notes in Computer Science*, pages 58–71, Busan, Korea, November 2007. Springer Verlag.

Karsten M. Borgwardt and Hans-Peter Kriegel. Shortest-Path Kernels on Graphs. In Jaiwei Han and Benjamin Wah, editors, *Proceedings of the Fifth IEEE International Conference on Data Mining (ICDM 2005)*, pages 74–81, Washington, DC, USA, November 2005. IEEE Computer Society.

Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A Training Algorithm for Optimal Margin Classifiers. In David Haussler, editor, *Proceedings of the 5th Annual Workshop on Computational Learning Theory (COLT'92)*, pages 144–152, Pittsburgh, PA, USA, July 1992. ACM Press.

Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, and François Yergeau. Extensible Markup Language (XML) 1.0 (Fifth Edition). W3C Recommendation, 2008. Available at `http://www.w3.org/TR/REC-xml/`.

Dan Brickley and Libby Miller. FOAF Vocabulary Specification. Technical report, FOAF project, May 2007. Published online on May 24th, 2007 at `http://xmlns.com/foaf/spec/20070524.html`.

Chris Buckley and Ellen M. Voorhees. Retrieval evaluation with incomplete information. In Mark Sanderson, Kalervo Järvelin, James Allan, and Peter Bruza, editors, *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2004)*, pages 25–32, Sheffield, United Kingdom, July 2004. ACM.

Paul Buitelaar, Daniel Olejnik, and Michael Sintek. A Protege Plug-In for Ontology Extraction from Text Based on Linguistic Analysis. In *Proceedings of the 1st European Semantic Web Symposium (ESWS)*, Heraklion, Greece, 2004.

Diego Calvanese, Evgeny Kharlamov, Werner Nutt, and Dmitriy Zheleznyakov. Evolution of DL-lite knowledge bases. In Peter F. Patel-Schneider, Yue Pan, Pascal Hitzler, Peter Mika, Lei Zhang, Jeff Z. Pan, Ian Horrocks, and Birte Glimm, editors, *Proceedings of the 9th International Semantic Web Conference (ISWC 2010)*, pages 112–128, Shanghai, China, 2010. Springer.

Eric Carle. *The Very Hungry Caterpillar*. Philomel Books, 1969.

Deepayan Chakrabarti and Christos Faloutsos. Graph mining: Laws, Generators, and Algorithms. *ACM Computing Surveys CSUR*, 38(1), 2006.

Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A Library for Support Vector Machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

Philipp Cimiano and Johanna Völker. Text2Onto - A Framework for Ontology Learning and Data-driven Change Discovery. In Andres Montoyo, Rafael Munoz, and Elisabeth Metais, editors, *Proceedings of the 10th International Conference on Applications of Natural Language to Information Systems (NLDB 2005)*, volume 3513 of *Lecture Notes in Computer Science*, pages pp. 227–238, Alicante, Spain, June 2005. Springer.

Philipp Cimiano, Andreas Hotho, and Steffen Staab. Learning Concept Hierarchies from Text Corpora using Formal Concept Anaylsis. *Journal of Artificial Intelligence Research (JAIR)*, 24:305–339, August 2005.

Philipp Cimiano, Sebastian Rudolph, and Helena Hartfiel. Computing Intensional Answers to Questions - An Inductive Logic Programming Approach. *Data and Knowledge Engineering*, 69(3):261–278, 2010.

Kendall Grant Clark, Lee Feigenbaum, and Elias Torres. SPARQL Protocol for RDF, 2008. Published online on January 15th, 2008 at `http://www.w3.org/TR/2008/REC-rdf-sparql-protocol-20080115/`.

Fabrizio Costa and Kurt De Grave. Fast Neighborhood Subgraph Pairwise Distance Kernel. In Johannes Fürnkranz and Thorsten Joachims, editors, *Proceedings of the 26th International Conference on Machine Learning (ICML 2010)*, pages 255–262, Haifa, Israel, June 2010. Omnipress.

Nello Cristianini and John Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, Cambridge, UK, March 2000.

Chad M. Cumby and Dan Roth. On Kernel Methods for Relational Learning. In T. Fawcett and N.Mishra, editors, *Proceedings of the 20th International Conference on Machine Learning (ICML 2003)*, pages 107–114, Washington, DC, USA, August 2003. AAAI Press.

Claudia d'Amato, Nicola Fanizzi, and Floriana Esposito. Query Answering and Ontology Population: an Inductive Approach. In Manfred Hauswirth, Manolis Koubarakis, and Sean Bechhofer, editors, *Proceedings of the 5th European*

Semantic Web Conference (ESWC 2008), volume 5021 of *Lecture Notes in Computer Science*, Tenerife, Spain, June 2008. Springer Verlag.

Christopher J. Date and Hugh Darwen. ISO/IEC 9075-2:2008 (SQL - Part 2: Foundations) , "The SQL Standard", Third Edition (Addison-Wesley Publishing Company, 1993), 2008.

Mukund Deshpande, Michihiro Kuramochi, Nikil Wale, and George Karypis. Frequent Substructure-Based Approaches for Classifying Chemical Compounds. *IEEE Transactions on Knowledge and Data Engineering*, 17:1036–1050, August 2005.

Paul D. Dobson and Andrew J. Doig. Distinguishing Enzyme Structures from Non-enzymes Without Alignments. *Journal of Molecular Biology*, 330(4):771 – 783, 2003.

Nicola Fanizzi and Claudia d'Amato. A Declarative Kernel for $\mathcal{ALC}$ Concept Descriptions. In Floriana Esposito, editor, *Proceedings of the 16th International Symposium on Methodologies for Intelligent Systems (ISMIS 2006)*, volume 4203 of *Lecture Notes in Artificial Intelligence*, pages 322–331, Bari, Italy, 2006. Springer.

Nicola Fanizzi and Claudia d'Amato. Inductive Concept Retrieval and Query Answering with Semantic Knowledge Bases Through Kernel Methods. In Bruno Apolloni, Robert J. Howlett, and Lakhmi C. Jain, editors, *Proceedings of the 11th International Conference on Knowledge-Based Intelligent Information and Engineering Systems (KES 2007)*, volume 4692 of *Lecture Notes in Artificial Intelligence*, pages 148–155, Vietri sul Mare, Italy, 2007. Springer.

Nicola Fanizzi, Claudia d'Amato, and Floriana Esposito. Induction of optimal semi-distances for individuals based on feature sets. In Diego Calvanese, Enrico Franconi, Volker Haarslev, Domenico Lembo, Boris Motik, Anni-Yasmin Turhan, and Sergio Tessaris, editors, *Working Notes of the 20th International Description Logics Workshop (DL 2007)*, volume 250 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007.

Nicola Fanizzi, Claudia d'Amato, and Floriana Esposito. Learning with Kernels in Description Logics. In Filip Železný and Nada Lavrac, editors, *Proceedings of the 18th International Conference on Inductive Logic Programming (ILP 2008)*, volume 5194 of *Lecture Notes in Computer Science*, pages 210–225, Prague, Czech Republic, September 2008a. Springer Verlag.

Nicola Fanizzi, Claudia d'Amato, and Floriana Esposito. Statistical Learning for Inductive Query Answering on OWL Ontologies. In Amit P. Sheth, Steffen Staab, Mike Dean, Massimo Paolucci, Diana Maynard, Timothy W. Finin,

and Krishnaprasad Thirunarayan, editors, *Proceedings of the 7th International Semantic Web Conference (ISWC 2008)*, volume 5318 of *Lecture Notes in Computer Science*, pages 195–212, Karlsruhe, Germany, October 2008b. Springer.

Christiane Fellbaum. WordNet and wordnets. In Keith Brown, editor, *Encyclopedia of Language and Linguistics*, pages 665–670, Oxford, 2005. Elsevier.

Ronald A. Fisher. The Use of Multiple Measurements in Taxonomic Problems. *Annals of Eugenics*, 7:179–188, 1936.

Giorgos Flouris, Dimitris Manakanatas, Haridimos Kondylakis, Dimitris Plexousakis, and Grigoris Antoniou. Ontology change: Classification and Survey. *The Knowledge Engineering Review*, 23(02):117–152, 2008.

Holger Fröhlich, Jörg K. Wegner, Florian Sieker, and Andreas Zell. Optimal assignment kernels for attributed molecular graphs. In Luc de Raedt and Stefan Wrobel, editors, *Proceedings of the 22nd International Conference on Machine Learning (ICML 2005)*, pages 225–232, Bonn, Germany, August 2005. ACM Press.

Bernhard Ganter and Rudolf Wille. *Formal Concept Analysis: Mathematical Foundations*. Springer, Berlin/Heidelberg, 1999.

Thomas Gärtner, John W. Lloyd, and Peter A. Flach. Kernels for Structured Data. In Stan Matwin and Claude Sammut, editors, *Proceedings of the 12th International Conference on Inductive Logic Programming (ILP'02)*, volume 2583 of *Lecture Notes in Computer Science*, pages 66–83, Sydney, Australia, July 2002. Springer Verlag.

Thomas Gärtner, Peter Flach, and Stefan Wrobel. On Graph Kernels: Hardness Results and Efficient Alternatives. In Bernhard Schölkopf and Manfred K. Warmuth, editors, *Learning Theory and Kernel Machines*, volume 2777 of *Lecture Notes in Computer Science*, pages 129–143. Springer, Berlin / Heidelberg, 2003.

Thomas Gärtner, John W. Lloyd, and Peter A. Flach. Kernels and Distances for Structured Data. *Machine Learning*, 57(3):205–232, 2004.

Lise Getoor and Christopher P. Diehl. Link mining: a survey. *ACM SIGKDD Explorations Newsletter*, 7(2):3–12, 2005.

Gerhard Goos. *Vorlesungen über Informatik Band 1: Grundlagen und funktionales Programmieren*. Springer, 3rd edition edition, January 2000.

Stephan Grimm and Jens Wissmann. Elimination of Redundancy in Ontologies. In Grigoris Antoniou, Marko Grobelnik, Elena Paslaru Bontas Simperl, Bijan

Parsia, Dimitris Plexousakis, Pieter De Leenheer, and Jeff Pan, editors, *Proceedings of the 8th Extended Semantic Web Conference (ESWC 2011)*, volume 6643 of *Lecture Notes in Computer Science*, pages 260–274. Springer-Verlag, 2011.

Ralf H. Güting. *Datenstrukturen und Algorithmen.* B.G. Teubner Stuttgart, 1992.

David Haussler. Convolution Kernels on Discrete Structures. Technical Report UCS-CRL-99-10, University of California at Santa Cruz, Santa Cruz, CA, USA, 1999.

Patrick Hayes. RDF Semantics. W3C Recommendation, 2004. Published online on February 10th, 2004 at http://www.w3.org/TR/2003/PR-rdf-mt-20031215/.

Sebastian Hellmann, Jens Lehmann, and Sören Auer. Learning of OWL Class Descriptions on Very Large Knowledge Bases. *International Journal on Semantic Web and Information Systems*, 5(2):25–48, 2009.

Pascal Hitzler, Markus Krötzsch, Bijan Parsia, Peter F. Patel-Schneider, and Sebastian Rudolph. OWL 2 Web Ontology Language Primer. W3C Recommendation, October 2009. Available at `http://www.w3.org/TR/2009/REC-owl2-primer-20091027/`.

Tamás Horváth, Thomas Gärtner, and Stefan Wrobel. Cyclic Pattern Kernels for Predictive Graph Mining. In Won Kim, Ron Kohavi, Johannes Gehrke, and William DuMouchel, editors, *Proceedings of the 10t ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2004)*, pages 158–167, Seattle, WA, USA, August 2004. ACM Press, New York, NY, USA.

Yi Huang, Volker Tresp, Markus Bundschus, Achim Rettinger, and Hans-Peter Kriegel. Multivariate Structured Prediction for Learning on Semantic Web. In Paolo Frasconi and Francesca A. Lisi, editors, *Proceedings of the 20th International Conference on Inductive Logic Programming (ILP 2010)*, volume 6489 of *Lecture Notes in Computer Science*, pages 92–104, Firenze, Italy, 2010. Springer Verlag.

Kalervo Järvelin and Jaana Kekäläinen. IR evaluation methods for retrieving highly relevant documents. In Emmanuel Yannakoudakis, Nicholas Belkin, Peter Ingwersen, and Mun-Kew Leong, editors, *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2000)*, pages 41–48, New York, NY, USA, 2000. ACM.

Thorsten Joachims. Making Large-Scale SVM Learning Practical. In Bernhard Schölkopf, Christopher John C. Burges, and Alexander J. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*, pages 169–184, Cambridge, MA, USA, 1999. MIT Press,.

Hirofumi Katsuno and Alberto O. Mendelzon. On the Difference between Updating a Knowledge Base and Revising it. In Peter Gärdenfors, editor, *Belief Revision*, pages 183–203. Cambridge University Press, December 1992.

Victoria Anna Kayser. Interaktive Ontologieupdates. Master's thesis, Karlsruhe Institute of Technology, May 2010.

Christoph Kiefer, Abraham Bernstein, and André Locher. Adding Data Mining Support to SPARQL Via Statistical Relational Learning Methods. In Manfred Hauswirth, Manolis Koubarakis, and Sean Bechhofer, editors, *Proceedings of the 5th European Semantic Web Conference (ESWC 2008)*, volume 5021, pages 478–492, Tenerife, Spain, 2008. Springer Verlag.

Michael Klein and Dieter Fensel. Ontology Versioning on the Semantic Web. In Isabel F. Cruz, Stefan Decker, JÃ©rÃ´me Euzenat, and Deborah L. McGuinness, editors, *Proceedings of the 1st International Semantic Web Working Symposium*, pages 75–91, Stanford University, CA, USA, July 2001.

Michel Klein, Dieter Fensel, Atanas Kiryakov, and Damyan Ognyanov. Ontology Versioning and Change Detection on the Web. In Asunción Gómez-Pérez and V. Richard Benjamins, editors, *Proceedings of the 13th European Conference on Knowledge Engineering and Knowledge Management (EKAW'02)*, pages 197–212, Siguenza, Spain, 2002.

Boris Konev, Dirk Walther, and Frank Wolter. Forgetting and Uniform Interpolation in Large-Scale Description Logic Terminologies. In Craig Boutilier, editor, *International Joint Conference on Artificial Intelligence (IJCAI 2009)*, pages 830–835, 2009.

George Konstantinidis, Giorgos Flouris, Grigoris Antoniou, and Vassilis Christophides. A Formal Approach for RDF/S Ontology Evolution. In Malik Ghallab, Constantine D. Spyropoulos, Nikos Fakotakis, and Nikolaos M. Avouris, editors, *Proceedings of the 18th European Conference on Artificial Intelligence (ECAI 2008)*, pages 70–74, Patras, Greece, 2008. IOS Press.

Taku Kudo, Eisaku Maeda, and Yuji Matsumoto. An Application of Boosting to Graph Classification. In Lawrence Saul, Yair Weiss, and Leon Bottou, editors, *Proceedings of the 18th Annual Conference on Neural Information Processing Systems (NIPS 2004)*, pages 729–736, Cambridge, MA, December 2004. MIT Press.

Jens Lehmann. Hybrid Learning of Ontology Classes. In Petra Perner, editor, *Proceedings of the 5th International Conference on Machine Learning and Data Mining in Pattern Recognition (MLDM 2007)*, volume 4571 of *Lecture Notes in Computer Science*, pages 883–898, Leipzig, Germany, July 2007. Springer Verlag.

Jens Lehmann. DL-Learner: Learning Concepts in Description Logics. *Journal of Machine Learning Research (JMLR)*, 10:2639–2642, 2009.

Jens Lehmann and Pascal Hitzler. A Refinement Operator Based Learning Algorithm for the ALC Description Logic. In Hendrik Blockeel, Jan Ramon, Jude W. Shavlik, and Prasad Tadepalli, editors, *Proceedings of the 17th International Conference on Inductive Logic Programming (ILP 2007)*, volume 4894 of *Lecture Notes in Computer Science*, pages 147–160. Springer, 2007.

Hector Levesque, Fiora Pirri, and Ray Reiter. Foundations for the Situation Calculus. *Electronic Transactions on Artificial Intelligence*, Vol. 2(1998), Issue 3-4:pp. 159–178, 1998.

David Liben-Nowell and Jon Kleinberg. The link prediction problem for social networks. In *Proceedings of the 12th International Conference on Information and Knowledge Management (CIKM 2003)*, pages 556–559, New Orleans, LA, USA, 2003. ACM.

Francesca A. Lisi and Floriana Esposito. Foundations of Onto-Relational Learning. In Filip Zelezný and Nada Lavrac, editors, *Proceedings of the 18th International Conference on Inductive Logic Programming (ILP 2008)*, volume 5194 of *Lecture Notes in Computer Science*, pages 158–175, Prague, Czech Republic, 2008. Springer.

Uta Lösch, Sebastian Rudolph, Denny Vrandečić, and Rudi Studer. Tempus Fugit - Towards an Ontology Update Language. In Lora Aroyo et al., editor, *6th European Semantic Web Conference (ESWC 09)*, volume 5554 of *Lecture Notes on Computer Science*, pages 278–292. Springer-Verlag, Juni 2009.

Uta Lösch, Stephan Bloehdorn, and Achim Rettinger. Graph Kernels for RDF Data. In Elena Simperl, Philipp Cimiano, Axel Polleres, Óscar Corcho, and Valentina Presutti, editors, *The Semantic Web: Research and Applications - 9th Extended Semantic Web Conference, ESWC 2012*, volume 7295 of *Lecture Notes in Computer Science*, pages 134–148. Springer Verlag, May 2012. Best Research Paper Award.

Carsten Lutz, Frank Wolter, and Michael Zakharyaschev. Temporal Description Logics: A Survey. In James Pustejovsky and Peter Revesz, editors, *Proceedings of the 15th International Symposium on Temporal Representation and*

Reasoning (TIME 2008), pages 3–14, Budapest, Hungary, June 2008. IEEE Computer Society.

Frank Manola and Eric Miller. RDF Primer. W3C recommendation, W3C, February 2004. Published online on February 10th, 2004 at `http://www.w3.org/TR/2004/REC-rdf-primer-20040210/`.

Giovanni Da San Martino and Alessandro Sperduti. Mining Structured Data. *IEEE Computational Intelligence Magazine*, 5(1):42–49, 2010.

Alistair Miles and Dan Brickley. SKOS Core Guide. W3C working draft, W3C, November 2005. Published online on November 2nd, 2005 at `http://www.w3.org/TR/2005/WD-swbp-skos-core-guide-20051102/`.

Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

Alessandro Moschitti. Efficient Convolution Kernels for Dependency and Constituent Syntactic Trees. In Johannes Fürnkranz, Tobias Scheffer, and Myra Spiliopoulou, editors, *Proceedings of the 17th European Conference on Machine Learning (ECML 2006)*, volume 4212 of *Lecture Notes in Computer Science*, pages 318–329, Berlin, Germany, September 2006. Springer Verlag.

Stephen Muggleton and Luc de Raedt. Inductive Logic Programming: Theory and methods. *The Journal of Logic Programming*, 19-20(0):629 – 679, 1994.

Nadeschda Nikitina, Sebastian Rudolph, and Birte Glimm. Reasoning-Supported Interactive Revision of Knowledge Bases. In Toby Walsh, editor, *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011)*, pages 1027–1032, 2011.

Heather S. Packer, Nicholas Gibbins, and Nicholas R. Jennings. Forgetting Fragments from Evolving Ontologies. In Peter F. Patel-Schneider, Yue Pan, Pascal Hitzler, Peter Mika, Lei Zhang, Jeff Z. Pan, Ian Horrocks, and Birte Glimm, editors, *Proceedings of the 9th International Semantic Web Conference (ISWC 2010)*, volume 6496 of *Lecture Notes in Computer Science*, pages 582–597. Springer Verlag, 2010a.

Heather S. Packer, Nicholas Gibbins, and Nicholas R. Jennings. Collaborative Learning of Ontology Fragments by Co-operating Agents. In Jimmy Xiangji Huang, Ali A. Ghorbani, Mohand-Said Hacid, and Takahira Yamaguchi, editors, *Proceedings of the 2010 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT 2010)*, pages 89–96, Washington, DC, USA, 2010b. IEEE Computer Society.

Vicky Papavassiliou, Giorgos Flouris, Irini Fundulaki, Dimitris Kotzinos, and Vassilis Christophides. On Detecting High-Level Changes in RDF/S KBs. In

Abraham Bernstein, David R. Karger, Tom Heath, Lee Feigenbaum, Diana Maynard, Enrico Motta, and Krishnaprasad Thirunarayan, editors, *Proceedings of the 8th International Semantic Web Conference (ISWC 2009)*, volume 5823 of *Lecture Notes in Computer Science*, pages 473–488, Washington, DC, USA, October 2009. Springer Verlag.

Eric Prud'hommeaux and Andy Seaborne. SPARQL Query Language for RDF. W3C Recommendation, 15 Jan. 2008. Available at `http://www.w3.org/TR/rdf-sparql-query/`.

Guilin Qi and Fangkai Yang. A Survey of Revision Approaches in Description Logics. In Diego Calvanese and Georg Lausen, editors, *Proceedings of the 2nd International Conference on Web Reasoning and Rule Systems (RR 2008)*, volume 5341 of *Lecture Notes in Computer Science*, pages 74–88, Karlsruhe, Germany, October 2008. Springer-Verlag.

Achim Rettinger, Matthias Nickles, and Volker Tresp. Statistical Relational Learning with Formal Ontologies. In Wray L. Buntine, Marko Grobelnik, Dunja Mladenic, and John Shawe-Taylor, editors, *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD 2009)*, volume 5782 of *Lecture Notes in Computer Science*, pages 286–301, Bled, Slovenia, September 2009. Springer, Berlin–Heidelberg, Germany.

Frank Rosenblatt. The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. *Psychological Reviews*, 65(6):386–408, November 1958.

Sebastian Rudolph. *Relational Exploration - Combining Description Logics and Formal Concept Analysis for Knowledge Specification.* Phd thesis, Dezember 2006.

Simon Schenk, Paul Gearon, and Alexandre Passant. SPARQL 1.1 Update. Technical report, W3C, 2008. Published online on October 14th, 2010 at `http://www.w3.org/TR/2010/WD-sparql11-update-20101014/`.

Bernhard Schölkopf, Alex J. Smola, and Klaus-Robert Müller. Nonlinear Component Analysis as a Kernel Eigenvalue Problem. Technical Report 44, Max Planck Institute for Biological Cybernetics, Tübingen, Germany, 1996.

Bernhard Schölkopf, Alex J. Smola, Robert C. Williamson, and Peter L. Bartlett. New Support Vector Algorithms. *Neural Computation*, 12(5):1207–1245, May 2000.

Bernhard Schölkopf, John C. Platt, John Shawe-Taylor, Alex J. Smola, and Robert C. Williamson. Estimating the Support of a High-Dimensional Distribution. *Neural Computation*, 13(7), July 2001.

Barış Sertkaya. OntoComP: A Protege Plugin for Completing OWL Ontologies. In *Proceedings of the 6th European Semantic Web Conference, (ESWC 2009)*, volume 5554 of *Lecture Notes in Computer Science*, pages 898–902. Springer Verlag, 2009.

John Shawe-Taylor and Nello Christianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.

Nino Shervashidze and Karsten Borgwardt. Fast Subtree Kernels on Graphs. In Yoshua Bengio, Dale Schuurmans, John Lafferty, Chris Williams, and Aron Culotta, editors, *Advances in Neural Information Processing Systems 22 – Proceedings of the 2001 Neural Information Processing Systems Conference NIPS 2009, December 7-10, 2009 Vancouver, British Columbia, Canada*, pages 1660–1668. Neural Information Processing Systems Foundation, 2009.

Nino Shervashidze, S. V. N. Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten M. Borgwardt. Efficient graphlet kernels for large graph comparison. *Journal of Machine Learning Research - Proceedings Track*, 5:488–495, 2009.

Leen-Kiat Soh. Multiagent Distributed Ontology Learning. In *Working Notes of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems – Workshop on Ontologies in Agent Systems*, Bologna, Italy, July 2002.

Ljiljana Stojanovic. *Methods and Tools for Ontology Evolution*. PhD thesis, Universität Karlsruhe, 2004.

Rudi Studer, V. Richard Benjamins, and Dieter Fensel. Knowledge engineering: Principles and methods. volume 25, pages 161–197, 1998.

York Sure, Steffen Staab, and Rudi Studer. Methodology for Development and Employment of Ontology Based Knowledge Management Applications. *SIGMOD Record*, 31:18–23, December 2002.

York Sure, Stephan Bloehdorn, Peter Haase, Jens Hartmann, and Daniel Oberle. The SWRC Ontology - Semantic Web for Research Communities. In Carlos Bento, Amilcar Cardoso, and Gael Dias, editors, *Proceedings of the 12th Portuguese Conference on Artificial Intelligence - Progress in Artificial Intelligence (EPIA 2005)*, pages 218–231, 2005.

Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining*. Addison Wesley, 2006.

Christoph Tempich, Elena Simperl, Markus Luczak, Rudi Studer, and Sofia H. Pinto. Argumentation-Based Ontology Engineering. *IEEE Intelligent Systems*, 22(6):52–59, November 2007.

Volker Tresp, Yi Huang, Markus Bundschus, and Achim Rettinger. Scalable Relational Learning for Sparse and Incomplete Domains. In Pedro Domingos and Kristian Kersting, editors, *International Workshop on Statistical Relational Learning (SRL 2009)*, 2009.

Tania Tudorache, Natasha F. Noy, Samson Tu, and Mark A. Musen. Supporting Collaborative Ontology Development in Protégé. In Amit P. Sheth, Steffen Staab, Mike Dean, Massimo Paolucci, Diana Maynard, Timothy W. Finin, and Krishnaprasad Thirunarayan, editors, *Proceedings of the 7th International Semantic Web Conference (ISWC2008)*, volume 5318 of *Lecture Notes in Computer Science*, pages 17–32, Karlsruhe, Germany, October 2008. Springer Verlag.

Vladimir Vapnik, Steven E. Golowich, and Alex J. Smola. Support Vector Method for Function Approximation, Regression Estimation and Signal Processing. In Michael Mozer, Michael I. Jordan, and Thomas Petsche, editors, *Advances in Neural Information Processing Systems 9 — Proceedings of the 1996 Neural Information Processing Systems Conference (NIPS 1996)*, pages 281–287, Dever, CO, USA, December 1997. MIT Press, Cambridge, MA, USA.

Vladimir N. Vapnik. *The Nature of Statistical Learning Theory*. Springer New York Inc., New York, NY, USA, 1995.

Paola Velardi, Roberto Navigli, Alessandro Cucchiarelli, and Francesca Neri. Evaluation of OntoLearn, a Methodology for Automatic Learning of Ontologies. In Paul Buitelaar, Philipp Cimiano, and Bernardo Magnini, editors, *Ontology Learning from Text: Methods, Evaluation and Applications*, pages 92–105. IOS Press, 2005.

S. V. N. Vishwanathan and Alexander J. Smola. Fast Kernels for String and Tree Matching. In Suzanna Becker, Sebastian Thrun, and Klaus Obermayer, editors, *Advances in Neural Information Processing Systems 15 — Proceedings of the 2002 Neural Information Processing Systems Conference (NIPS 2002)*, pages 569–576, Vancouver, British Columbia, Canada, December 2003. MIT Press, Cambridge, MA, USA.

Max Völkel and Tudor Groza. SemVersion: An RDF-based Ontology Versioning System. In Miguel Baptista Nunes, editor, *Proceedings of IADIS International Conference on WWW/Internet (IADIS 2006)*, pages 195–202, Murcia, Spain, October 2006.

Johanna Völker and Mathias Niepert. Statistical schema induction. In Marko Grobelnik and Elena Simperl, editors, *Proceedings of the 8th Extended Semantic Web Conference (ESWC 2011)*, ESWC'11, pages 124–138, Heraklion, Crete, 2011. Springer-Verlag.

Johanna Völker and Sebastian Rudolph. Lexico-logical Acquisition of OWL DL Axioms: an Integrated Approach to Ontology Refinement. In Raoul Medina and Sergei Obiedkov, editors, *Proceedings of the 6th International Conference on Formal Concept Analysis (ICFCA 2008)*, pages 62–77, Montreal, Canada, 2008. Springer.

Johanna Völker, Pascal Hitzler, and Philipp Cimiano. Acquisition of OWL DL Axioms from Lexical Resources. In Enrico Franconi, Michael Kifer, and Wolfgang May, editors, *Proceedings of the 4th European Semantic Web Conference (ESWC 2007)*, volume 4519 of *Lecture Notes in Computer Science*, pages 670–685, Innsbruck, Austria, 2007a. Springer.

Johanna Völker, Denny Vrandecic, York Sure, and Andreas Hotho. Learning Disjointness. In Wolfgang May Enrico Franconi, Michael Kifer, editor, *Proceedings of the 4th European Semantic Web Conference (ESWC 2007)*, volume volume 4519 of *Lecture Notes in Computer Science*, pages 175–189, Innsbruck, Austria, Juni 2007b. Springer.

Denny Vrandečić. *Ontology Evaluation*. PhD thesis, Karlsruhe Institute of Technology, 2010.

Kewen Wang, Zhe Wang, Rodney Topor, Jeff Z. Pan, and Grigoris Antoniou. Concept and Role Forgetting in ALC Ontologies. In Abraham Bernstein and David Karger, editors, *Proceedings of the 8th International Semantic Web Conference (ISWC2009)*, volume 5318 of *Lecture Notes in Computer Science*, October 2009.

Zhe Wang, Kewen Wang, Rodney W. Topor, and Jeff Z. Pan. Forgetting Concepts in DL-Lite. In Sean Bechhofer, Manfred Hauswirth, Jörg Hoffmann, and Manolis Koubarakis, editors, *Proceedings of the 7th European Semantic Web Conference (ESWC 2008)*, volume 5021 of *Lecture Notes in Computer Science*, pages 245–257. Springer, June 2008.

Zhao Xu, Volker Tresp, Kai Yu, and Hans Peter Kriegel. Infinite Hidden Relational Models. In *Proceedings of the 22nd International Conference on Uncertainity in Artificial Intelligence (UAI 2006)*, Cambridge, MA, USA, July 2006. AUAI Press.

Xifeng Yan and Jiawei Han. gSpan: Graph-Based Substructure Pattern Mining. In *Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM 2002)*, pages 721–724, Maebashi City, Japan, December 2002. IEEE Computer Society.

Zhijun Yin, Manish Gupta, Tim Weninger, and Jiawei Han. LINKREC: A Unified Framework for Link Recommendation with User Attributes and Graph

Structure. In *Proceedings of the 19th International Conference on World Wide Web (WWW 2010)*, WWW '10, pages 1211–1212, New York, NY, USA, 2010. ACM.