Supervised and Unsupervised Ensemble Learning for the Semantic Web

Andreas Heß

A thesis submitted to the National University of Ireland, Dublin for the degree of Doctor of Philosophy in the Faculty of Science

February, 2006

School of Computer Science and Informatics
National University of Ireland, Dublin
Belfield, Dublin 4, Ireland

Head of Department: Barry Smyth
Supervisor: Nicholas Kushmerick

Contents

List of Figures

List of Tables

List of Algorithms

Summary

The World Wide Web offers an ocean of information and services for everyone, and many of us rely on it on a daily basis. One of the reasons for the Web's success is that the threshold for creating content on the Web was always very low, because comfortable and easy to use HTML editors are available. Today, so many Web pages exist that it is hard to filter out the information that is actually relevant. This problem is known as "information overload". Conventional search engines are limited by the fact that the Web is *machine-readable*, but not *machine-understandable*. Intelligent text processing algorithms have been proposed that often work well for certain tasks, but do not scale to the size of the Web. The *Semantic Web* promises a solution to this problem by adding explicit semantics with the goal of making the Web machine-understandable by using description logics and ontologies. However, the threshold for creating this extra markup is very high, and no comfortable tools exist at present. The goal of this work is to develop such tools. Machine Learning techniques have been used in literature for various classification tasks. The central claim of this thesis is that such algorithms, supervised and unsupervised, can be used for this purpose.

The goal as sketched here is split into several parts. First, the problem of assigning an appropriate category to a Web Service as a whole is discussed. The problem can be treated as supervised text classification. We show that a multi-view approach performs better than using a single classifier. We also apply an unsupervised clustering algorithm to the same problem and show that this approach is also feasible, although it is less accurate. Classifying services as a whole can be useful for UDDI or for mediating between different taxonomies. However, to assist the user in annotating Semantic Web Services, all components within a service need to be classified. We present a supervised iterative relational learning algorithm to address this problem and introduce the approach of specialised classifiers for relational learning.

Next, we generalise the problem from annotating Web Services to ontology mapping. We present an unsupervised iterative relational algorithm for graph matching, apply it to the ontology mapping problem and show that it outperforms several algorithms from recent literature. As a diversion, we present a new ensemble algorithm for general supervised learning tasks that we call Triskel. Although not specifically targeted at Semantic Web applications, Triskel was developed as a direct consequence of the specialised classifiers for relational learning.

Publications

The following research papers were prepared and published during the course of this work:

- Andreas Heß and Nicholas Kushmerick. "Automatically Attaching Semantic Metadata to Web Services", in *Proceedings of the IJCAI-03 Workshop on Information Integration on the Web (IIWeb-03)*, August 9-10, 2003, Acapulco, Mexico. pp. 111–116
- Andreas Heß and Nicholas Kushmerick. "Learning to Attach Semantic Metadata to Web Services", in *Proceedings of the Second International Semantic Web Conference*, October 20–23, 2003, Sanibel Island, Florida, USA. Lecture Notes in Computer Science, Vol. 2870, Springer, Germany. pp. 258–273
- Andreas Heß and Nicholas Kushmerick. "Machine Learning for Annotation Semantic Web Services", in *AAAI Spring Symposium Semantic Web Services*, March 22-24, Stanford, California, USA.
- Andreas Heß, Eddie Johnston and Nicholas Kushmerick. "Semi-Automatically Annotating Semantic Web Services", in *Proceedings of the VLDB-04 Workshop on Information Integration on the Web (IIWeb-04)*, August 30, 2004, Toronto, Canada.
- Andreas Heß and Nicholas Kushmerick. "Iterative Ensemble Classificatin for Relational Data: A Case Study of Semantic Web Services", in *Proceedings of the 15th European Conference on Machine Learining*, September 20–24, Pisa, Italy. Lecture Notes in Computer Science, Vol. 3201, Springer, Germany. pp. 156–167
- Andreas Heß, Eddie Johnston and Nicholas Kushmerick. "ASSAM: A Tool for Semi-automatically Annotating Semantic Web Services", in *Proceedings of the Third International Semantic Web Conference*, November 7–11, 2004, Hiroshima, Japan. Lecture Notes in Computer Science, Vol. 3298, Springer, Germany. pp. 320–334
- Andreas Heß, Eddie Johnston and Nicholas Kushmerick. "Machine Learning Techniques for Annotating Semantic Web Services", in *Proceedings of the Dagstuhl Seminar "Machine Learning for the Semantic Web"*, February 13–18, 2005, Schloss Dagstuhl, Germany.

- Andreas Heß, Rinat Khoussainov and Nicholas Kushmerick. “Ensemble Learning with Biased Classifiers: The Triskel Algorithm”, in *Proceedings of the 6th International Workshop on Multiple Classifier Systems*, June 13–15, 2005, Seaside, California, USA. Lecture Notes in Computer Science, Vol. 3541, Springer, Germany. pp. 226–256

- Rinat Khoussainov, Andreas Heß and Nicholas Kushmerick. “Ensembles of Biased Classifiers”, in *Proceedings of the 22nd International Conference on Machine Learning”*, August 7–11, 2005, Bonn, Germany.

Acknowledgements

Many people deserve that I thank them for various things, and any attempt to enumerate them all must be futile. But I will give it a try anyway.

First, I would like to thank Nicholas Kushmerick for his support and advice. Our weekly meetings have always been a great source of inspiration. There could not have been a better PhD supervisor. Thanks go to my former officemates Rinat Khoussainov, Aidan Finn, John Meade, Eddie Johnston, Greg Murdoch, David Masterson, Brian McLernon, Frank McCarey, Mark O'Keeffe and Andrea Rizzini. Thanks go to Martina Naughton, Jessica Kenny, Wendy McNulty and again Andrea Rizzini for creating the ASSAM dataset. Thanks also to all other PhD students and staff members at the school of computer science and informatics. Hugs go to Bianca Schön, who came with me to Ireland. She helped me with various things. Among that, she created the dataset for the multi-view learning experiments and also proofread this thesis.

My new colleagues at the Vrije Universiteit Amsterdam, especially Frank van Harmelen and Annette ten Teije, deserve special thanks for allowing me to finish my thesis while getting starting on a new project. Thanks also to my new officemates Michel Klein and Radu Serban and to all others from the AI group for their warm welcome. Marta Sabou, who just left the group in Amsterdam, deserves thanks for interesting discussions. Thanks to many anonymous reviewers of our papers for valuable comments. Science Foundation Ireland and the US Office of Naval Research deserve thanks for providing the funding that made this research possible.

I would also like to thank all the people who were involved in the chain of events that led me to Dublin. Frank Hüllenhütter was one of the people at the start of the chain. The bosses at H.A.S.E, Wolfgang Grund, Dirk Masera and Hilmar Lenz, provided the best workplace that I could have wished for. Thanks also go to my professors at the FH Darmstadt, especially Prof. Arz and Prof. Kestner.

Thanks to my friends and family in Germany, who are still there when I need them, and to my new friends in Dublin. Special thanks go to Götz Schwandter for proofreading – especially the mathematical parts of this thesis – and for giving helpful comments, and to Thomas Gottron for explaining me some statistics. Among my family members, my uncle deserves special thanks for driving me to and from the airport on a countless number of occasions. And last but not least I want to thank my grandmother. Without her, this thesis would not exist. I dedicate it to her.

Declaration

I declare that this thesis is my own work and has not been submitted in any form for another degree or diploma at this, or any other, University or institute of tertiary education.

Andreas Heß
February 2, 2006

Chapter 1

Introduction

The World Wide Web offers an ocean of information and services for everyone. Although it is merely fifteen years old, many of us could not imagine life without it any more. One of the reasons for its success, beside its usefulness, is the fact that it is very easy for everyone to create a web page. HTML, the page description language of the Web, is very simple to learn. And, what is even more important, right from the beginning, comfortable and easy to use HTML editors existed.

By today, the Web has grown so large that we often face a problem of "information overload": In the mass of web pages that exist, how can we sift out the information that we were actually looking for? One of the drawbacks here is that HTML is a description language for the visual layout of a page and has no semantics. The Web is meant for human consumption. It is *machine-readable*, but not *machine-understandable*. To overcome this problem, information extraction techniques that use intelligent text processing to retrieve pieces of interest from a mass of often unstructured text have been proposed. While modern information extraction algorithms work reasonably well for certain tasks, this is a method that does not scale to the size of the Web.

The *Semantic Web* promises a solution to this problem. The idea of the Semantic Web is to make the implicit semantics that is in the text we find in the Web explicit, and thus meaningful for an algorithm. The vision is that we can use intelligent agents (e.g. (Hendler & Lassila 2001)) to go on the Web for us and autonomously perform a task that they were told to do. It is of course still a long way to go until we achieve this goal. But yet, this idea is so appealing that Tim Berners-Lee, the author of the first Web browser and often called the father of the World Wide Web, is actively promoting the Semantic Web (e.g. (Berners-Lee, Hendler & Lassila 2001)).

One of the main obstacles, if not *the* obstacle, the Semantic Web faces is that as opposed to the World Wide Web this annotation for the Semantic Web is not yet easy to create for everyone. The threshold for the average user to create a Web page is so low that millions of people create Web pages in their free time. The threshold for creating the extra piece of information that we need in the Semantic Web is so high that not even for-profit programmers bother to invest this extra effort, although this effort could ease data integration problems. The goal of this thesis is to lower the threshold

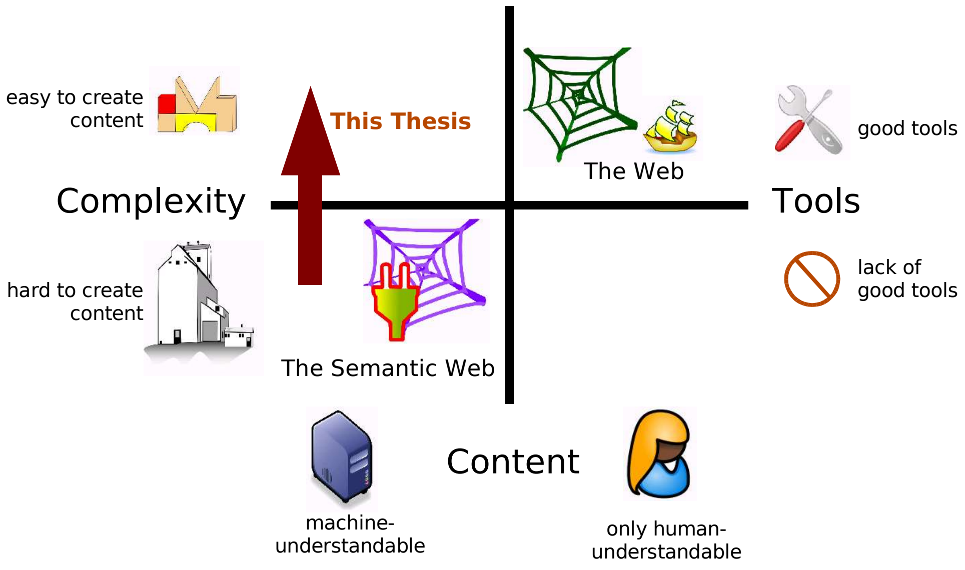


Figure 1.1: *The objective of this thesis is to simplify the creation of content on the Semantic Web by providing good tools.*

for creating semantic metadata significantly (see figure 1.1). We believe that machine learning techniques can be used to build tools that assist the user. In this thesis, we will show several approaches where we successfully apply both supervised and unsupervised ensemble learning to Semantic Web and data integration problems. Other approaches from literature are referred to and discussed in the respective chapters.

1.1 Motivation

Ever since the first databases, programmers have faced the problem of exchanging data between different systems. On many mainframe systems, this problem of data exchange or *data integration* is usually solved by defining a fixed position and length for each piece of data or field. This way of exchanging data is very efficient, but it has several disadvantages. Without adequate documentation, it is impossible for a programmer to guess either the meaning of each field, or even where a field starts or ends. If the data structure changes, all programs that participate in the communication have to be changed as well in order to work properly.

1.1.1 XML

With the advent of XML (Extensible Markup Language)[1], the hope was to ease some of these issues by creating a format that is to a certain extent self-describing. Tag names, if chosen sensibly, tell the programmer about the meaning and purpose of each field. If new fields are inserted into the data structure, older programs that are not aware of the new data can choose to just ignore the new tags. With Document Type Definitions (DTDs)[2]

[1] `http://www.w3.org/XML/`
[2] The specification for DTD is part of the XML specification.

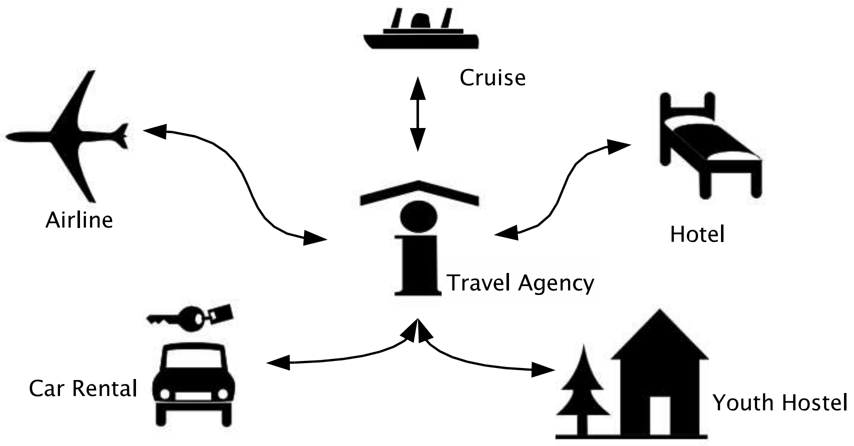


Figure 1.2: *An example for data integration with Web Services*

or the more sophisticated XML Schema[3] it is possible to formally verify the correctness of a document. With XSLT (Extensible Stylesheet Language Transformations)[4] it is possible to transform data from one schema to another. However, the "marketing promise" that XML makes everything compatible can of course not be fulfilled: This is only true as long as a mapping between different schemata is known.

In this context – and before in the context of relational databases – various approaches to (semi-) automatically generate such mappings have been proposed, for example Cupid (Madhavan, Bernstein & Rahm 2001), COMA (Do & Rahm 2002), Clio (Popa, Velegrakis, Miller, Hernandez & Fagin 2002), LSD (Doan, Domingos & Halevy 2003) and ILA (Perkowitz & Etzioni 1995).

1.1.2 Web Services

Based on XML, Web Services[5] promise an answer to the data integration problem not from a representational but rather from a transactional perspective. Consider a travel agency scenario as in figure 1.2. The customer who comes to the travel agency expects that he can book all parts of his trip, from the air ticket to the hotel and the rented car, all at once without having to contact the individual airlines, hotels and car rental agencies directly. The travel agent therefore needs to integrate the different services offered. This is a situation where Web Services become useful. Web Services are programs that are made available over the web via a standardised software interface. As opposed to (human-readable) web pages, this allows for an easy integration of third-party information (e.g. from the hotel or the airline) into a software system (e.g. at the travel agency).

The features that distinguish Web Services from older techniques for remotely exe-

[3]`http://www.w3.org/XML/Schema`
[4]`http://www.w3.org/TR/xslt`
[5]`http://www.w3.org/2002/ws/`

cuting code, such as the Common Object Request Broker Architecture (CORBA)[6] or the proprietary Distributed Component Object Model (DCOM) by Microsoft, is the use of XML both as a message exchange format and a description format, and the accessibility through firewalls because of the use of HTTP as the access method.

The interface of a web service is usually described in WSDL, the Web Service Description Language[7]. UDDI (Universal Description, Discovery and Integration)[8] promises an easy way of dynamically discovering web services.

However, UDDI suffers from some shortcomings. Although UDDI specifies a "Yellow Pages"-like service to discover web services according to a taxonomy, its search capabilities are very limited. Different competing taxonomies like the UNSPSC (United Nations Standard Products and Services Code) or NAICS (North American Industry Classification System) exist, and it is worth noting that these taxonomies were made for classifying industries, not web services. It is only possible to search for exact categories. Even the search for more general or more specific classifications is not possible. This means that, for example, if the query asks for a "finance" web service, a UDDI search will not result in web services classified as the more specific "banking". To make it even worse, a web service provider who advertises in a UDDI registry does not have to specify a code at all.

1.1.3 Ontologies

Tom Gruber defines an ontology as a formal "specification of a conceptualisation" (Gruber 1993). A slightly simplified view is that an ontology describes concepts and relations between these concepts. However, the purpose that ontologies usually fulfil, is that they are a well defined vocabulary that can be used to make assertions about the world. The role of ontologies in data exchange is very similar to the role of a schema, and so are the challenges that are associated with the use of ontologies.

As with schemas, the usefulness of ontologies for the purpose of data exchange and integration stands and falls with the ability of finding mappings between different ontologies. Again, ontologies are to some extent associated with the hope to ease the data integration problem. What XML (Schema) contributed towards the solution of this problem was to provide a formal self-description of the data. While XML allows a verification of the syntax, ontologies contribute formal assertions about the data itself.

1.1.4 Ontologies for Web Services

To facilitate automated *discovery, invocation and composition* of web services, a semantic, not only a syntactic description is needed.

Ontologies for Web Services such as OWL-S promise a solution to this problem by providing "a core set of markup language constructs for describing the properties and

[6]`http://www.corba.org/`
[7]`http://www.w3.org/TR/wsdl`
[8]`http://www.uddi.org/`

capabilities of their Web services in unambiguous, computer-interpretable form" (The DAML Services Coalition 2003). Although OWL-S provides a framework, it is by itself not enough to attach meaning to the elements of a Web Service, such as its parameters or even the category of the service as a whole. It is only useful if it is linked to a domain ontology. Selecting appropriate concepts from this domain ontology is the key challenge here.

A Web Service as modelled in OWL-S typically consists of several parts:

- Service Profile
- Service Model
- Grounding
- Concepts

The Service Profile is a description of what the Service does. It should be modelled so that it is sufficient for discovering the service. The Service Model is a description of how the Service works. It is usually more detailed than the profile and needed if the Service is to be invoked and composed with other Services. The Grounding ties the Service description as given in OWL-S to the WSDL. It defines how to invoke the Service syntactically. The above files can make use of new classes as defined in an ontology. To be useful, these concepts should be linked to a shared domain ontology. In addition to these aspects of OWL-S, usually an instance of class "Service" is declared that *presents* the profile, *is described by* the model and *supports* the grounding. "presents", "isDescribedBy" and "suppoorts" are in OWL-S modelled as properties of "Service".

1.2 Outline

The work that led to this thesis tried to tackle some of the problems arising from the need of mapping between different schemata, ontologies and Web Services as mentioned above by using machine learning techniques. Figure 1.3 show the topics covered by this thesis and their relations.

We begin by giving short introductions on ontologies (section 2.1) and Web Services, especially the Web Service Description Language WSDL (section 2.2), before we continue with some definitions (section 2.3) that serve as a prerequisite for the main chapters. After that, we give a very brief overview of machine learning (section 2.4).

1.2.1 Categorising Web Services

In the main part of the thesis, we consider the problem of assigning an appropriate category to the Web Service as a whole first. This problem is discussed in chapter 3. We start by specifying the problem (section 3.1) and presenting our corpus of Web Services (section 3.2). We found out that a multi-view approach where we use separate bag of

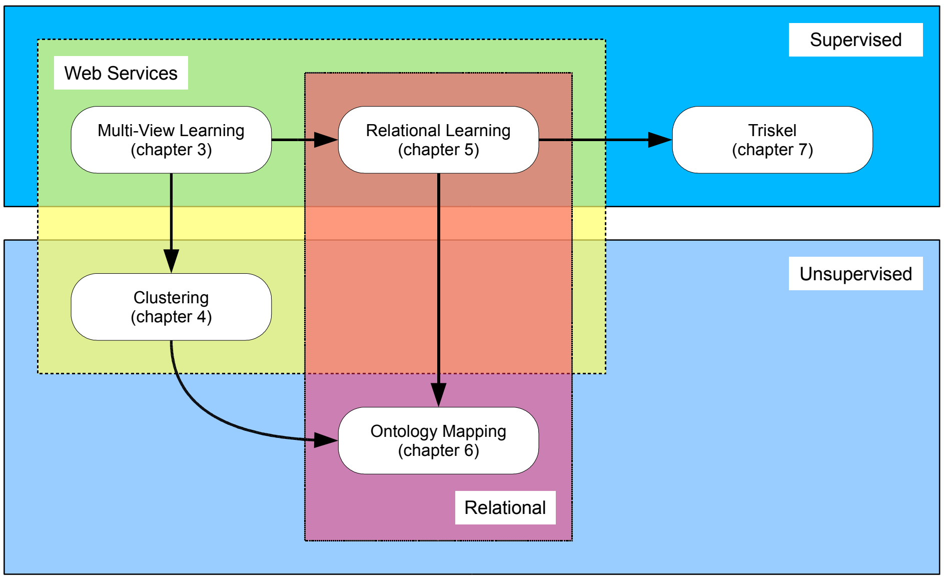

Supervised
Web Services
Multi-View Learning (chapter 3)
Relational Learning (chapter 5)
Triskel (chapter 7)
Unsupervised
Clustering (chapter 4)
Ontology Mapping (chapter 6)
Relational

Figure 1.3: *Outline of this thesis*

words for separate components of the Web Service and combine the results performs better than using a single classifier.

1.2.2 Clustering Web Services

We also tried unsupervised clustering (chapter 4) on the level of Web Services as a whole. We introduce a variant of standard hierarchical agglomerative clustering that produces concise labels for each cluster. Although the unsupervised method is, as expected, less accurate than the supervised approach, we have to conclude that both the supervised classification and unsupervised clustering of Web Services into categories are feasible.

1.2.3 Annotating Web Services

Classifying the category of a Web Service can already be useful when considering a scenario where we have to find an appropriate class in a taxonomy for use within UDDI or when we have to mediate between different competing taxonomies (see above). However, the main reason for doing this relatively straightforward work that treats Web Service classification as similar to text classification, was to test if the usually quite short texts that can be extracted from names and comments in Web Services are enough to make classification techniques work. The positive results obtained encouraged us to move on to the more complex task of classifying the inner components of a Web Service as well. This is of paramount importance, if we want to assist a human annotator in creating a Semantic Web Service.

Chapter 5 covers our approach to the problem of annotating the complete Web Service. In section 5.1 we present the ASSAM Annotator application, a prototypical software for adding semantics to Web Services. It assists a human annotator by suggesting possible annotations that are based on predictions obtained from a classifier. We use supervised iterative relational learning to make these predictions. Section 5.2 presents this technique in greater detail. In this section, we also introduce the notion of specialised classifiers for relational learning.

1.2.4 Aligning Ontologies

As the unsupervised annotation of Web Services can be regarded as merely a special case of the more generic problem of mapping a schema (the Web Service) to an ontology, we generalised our approach to ontology mapping. Chapter 6 discusses this generalisation. As in the preceding chapters, we begin in section 6.1 with a formulation of the problem of ontology mapping, frequently also called ontology alignment. In section 6.2 we present related work. Our algorithm for ontology mapping is, as the algorithm presented in chapter 5, an iterative relational algorithm. It is presented in detail in section 6.3. In section 6.4 we discuss the relation between ontology alignment and graph matching and show that the accuracy of an ontology alignment algorithm can be increased by applying an algorithm that searches for the maximum weighted matching.

1.2.5 The Triskel Algorithm

Before we come to the conclusion, in chapter 7 we present an ensemble learning algorithm that we call Triskel. We show that Triskel outperforms well-known established ensemble methods such as Boosting. Unlike the other approaches that have been discussed in this thesis so far, Triskel is not an algorithm for matching problems but a general machine learning algorithm. Chapter 7 is therefore a diversion. However, it was developed as a consequence of the work the specialised classifiers introduced in section 5.2. Because of that and because we think that Triskel is an interesting contribution to the field of ensemble learning, we discuss it here in spite of its relevance being beyond the specific problem domain with which this thesis is primarily concerned.

Chapter 2

Background

This chapter presents some background information on ontologies, Web Services and machine learning. These introductions are by no means complete. Their purpose is solely to enable a reader not familiar with one of these topics to follow the remainder of the thesis. In this chapter, we also introduce the mathematical notation that will be used throughout the thesis.

2.1 Introduction to Ontologies

2.1.1 The Semantic Web

The conventional World Wide Web, based on HTML, is meant for visual presentation and human consumption. Consider a typical web shop. For a human, it is easy to see which part of the web site is the title of a product, which is a description and which is the price. However, it is not so easy for a machine to, for example, extract all product names together with the price and transfer them into a table. The reason for that is that usually the markup in the presented document is purely visual. A program only sees text that is set in boldface, not what this piece of text actually *means*. For this extraction problem, it is, nevertheless, relatively easy to construct wrappers that extract certain information like the price (e.g. (Freitag & Kushmerick 2000, Muslea, Minton & Knoblock 1999)) However, this requires considerable human interaction, as the algorithm needs to be trained first. It is also error-prone, since the webmaster might change the structure of the website. Then, further attempts to extract new data using the same wrapper might fail, leading to no or, even worse, faulty data to be extracted.

In this scenario, the use of *information extraction* techniques is, however, just a (computationally expensive!) way to circumnavigate the fact that an explicit markup is not present. One of the goals of the semantic web is to provide such markup, thus making the web *machine understandable*.

2.1.2 Ontologies

Semantic markup alone does, however, not solve the problem of machine-understandability yet. Assume the task is to retrieve news articles about Ireland. Assuming there exists some geographic semantic markup, the task seems straightforward. But what if the markup in some of the articles states that it is about "Dublin"? For a human, it is easy to see that the article is still relevant.[1] This is possible because we can reason about known facts. For an algorithm, however, it is necessary to explicitly state that Dublin is in fact the capital of Ireland. Therefore it is clear that for the semantic web to work there is a need for something more than just a pure taxonomy.

By using an ontology, it is possible to not only arrange a lexicon of terms or concepts in hierarchy (as in a taxonomy), but also to issue formal statements about the relationship between concepts. For example, an ontology could state that Dublin is a city in Ireland, then we can infer that we should retrieve the news article, although it does not explicitly mention Ireland as a topic.

Unfortunately, there is no "global ontology" that would enable us to universally identify all entities in the real world. Rather, for many applications, domain ontologies, that cover a certain, usually very specific, range of expertise have been devised. Besides these various "bottom-up" ontologies, there are also attempts to create "upper" ontologies in a top-down way. Examples for such ontologies are OpenCyc[2] or the Suggested Upper Merged Ontology (SUMO)[3].

2.1.3 RDF, RDFS and OWL

The Resource Description Framework (RDF) is the basis for most widely used ontology languages currently used. It is based on *statements* about *resources* that are triples consisting of a *subject*, a *predicate* and an *object*. A resource in RDF is identified by a Uniform Resource Identifier (URI). While the subject and predicate of an RDF statement are always resources, the object can be either a resource or a literal (text). The predicate is a resource that identifies the relationship between the subject and the object.

RDF Schema or RDFS is an extension to RDF that allows for formal assertions about RDF statements. RDF Schema introduces the notion of classes, properties and instances.

The Web Ontology Language OWL (not WOL!) is a further extension. Historically it is based on DAML+OIL (which in turn is based on the DARPA agent markup language DAML, developed with funding from the US Department of Defense, and the Ontology Inference Layer OIL). OWL is much more expressive than RDFS and has a direct link to Description Logics. The variant OWL-DL has the same expressiveness as the $\mathcal{SHOIN}(D)$ description logics. The less expressive OWL Lite has the same ex-

[1]Unless it is about Dublin, Ohio.
[2]`http://www.cyc.com/cyc/opencyc/overview`
[3]`http://www.ontologyportal.org/`

pressiveness as the $\mathcal{SHIF}(D)$ description logics. The variant OWL-Full is even more expressive than OWL-DL, but reasoning about OWL-Full is undecidable.

This section briefly introduces some terms from ontologies as far as they are relevant to this thesis, but is not an exhaustive description.

Classes

A class in an ontology represents something that we can make assertions about. A class in an ontology usually corresponds to an entity in the real world, but can also represent some abstract concept. A class is sometimes also referred to as a concept or a category. Typically, a subsumption relation ("inheritance") is defined on classes. For example, consider a class "car" that is a subclass of "vehicle".

Instances

Instances or individuals are representations of concrete objects. The relation between classes and instances in ontologies is the same as between classes and objects in object oriented programming languages. An instance belongs to a class. For example, "TheMercedes" is an instance of class "car", and "Janis" is an instance of a "person".

Properties

Attributes or properties are descriptions of an instance. Properties can be declared on the level of classes and have concrete values on the level of instances. For example, we can define that the class "vehicle" has a property "owner", and that the "owner" of "TheMercedes" is "Janis".

We have to distinguish between *object* and *datatype* properties. An object property describes the relation between two instances or two classes. The property "owner" from our example is an object property. Its *domain* is "vehicle", and its range is "person", meaning that every vehicle can have an owner, and that an owner is always a person. A datatype property is a property that has a simple datatype like "integer" or "string" as its range. For example, we can declare a datatype property "lengthInCentimeters" with the range "integer" and the domain "vehicle".

Properties can also describe the relations between classes itself. For example, "subclassOf" and "superclassOf" (the subsumption relation on classes) are properties of a class. There is also a subsumption relation defined on properties. A property can be a subproperty of another property.

Figure 2.1 shows a graphical representation of a simple ontology. Classes are denoted by rectangles, properties are denoted by ellipses and instances are denoted by diamond shapes.

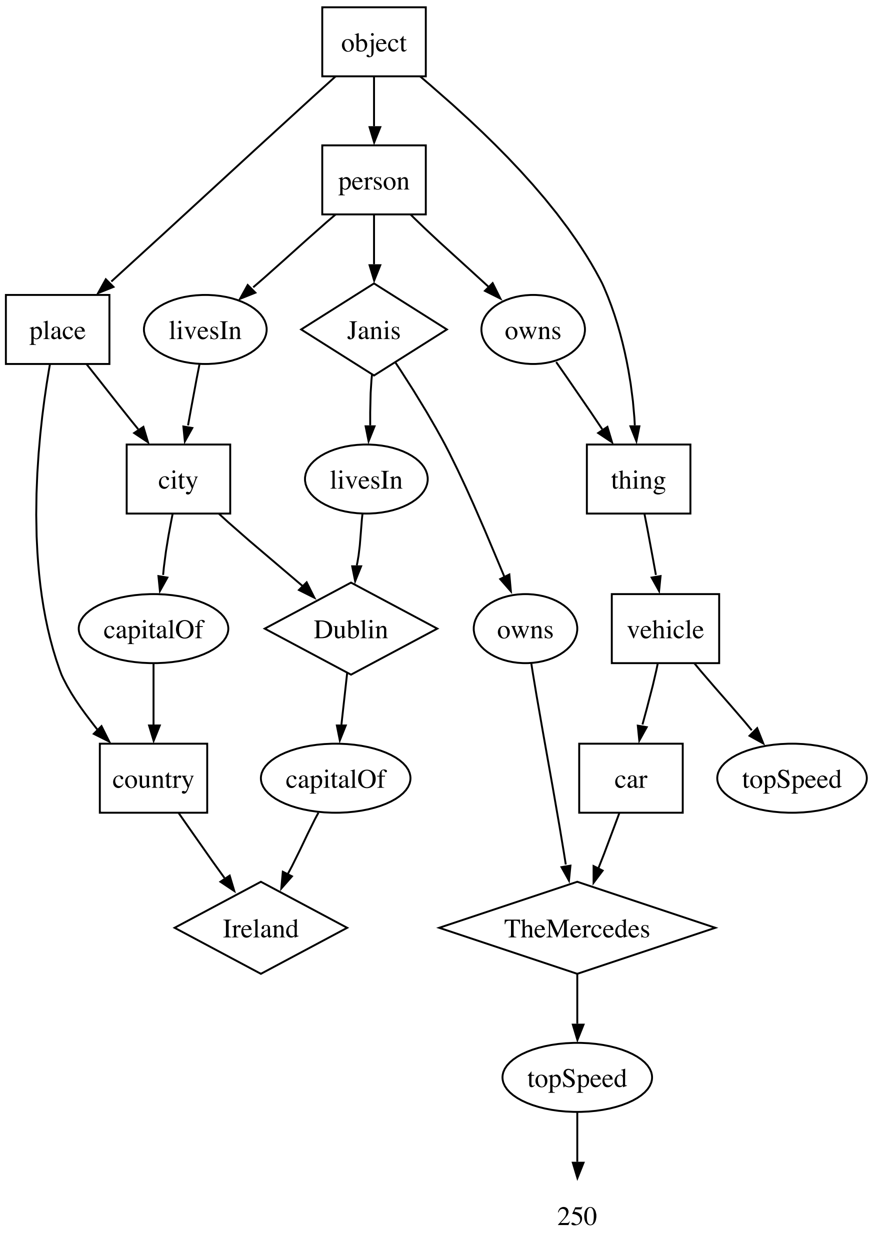


Figure 2.1: *Example Ontology*

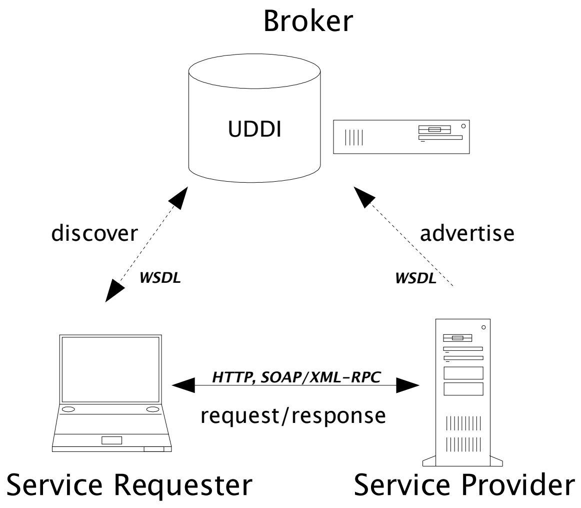


Figure 2.2: *Service Oriented Architecture*

2.1.4 Domain and Upper Ontologies

2.2 Introduction to Web Services

2.2.1 Web Service Protocol Stack

Web Services as loosely coupled software components play a key role in a *service oriented architecture*. The service oriented architecture (SOA) is an architectural concept that is based on self-contained services that are *composed* to implement complex business processes. This implies that the business logic is distributed over independent services. To make use of services, the SOA model allows a *service consumer* or *requester* to make *service requests* to a *service provider* and get a *service response*. To *discover* a service, the requester can query a *broker* or *registry*. This setup is illustrated in figure 2.2.

According to the requirements of the SOA, the protocol stack for Web Services covers transport, messaging, service description and service discovery. The most commonly used transport protocol is currently HTTP, but it is also possible to use SMTP or FTP. The most common messaging protocols, based on XML, are XML-RPC and SOAP. The Web Service Description language WSDL is commonly used for the syntactic description of a service, and UDDI is used for discovery.

The motivation of the work presented in this thesis lies in the need for a greater degree of automation in service discovery and composition. As shortly explained above, UDDI has only very limited discovery capabilities, and WSDL only provides a syntactic description of a Web Service. A human developer must carefully select suitable services in order to compose services in order to implement a business process. Ontologies for Web Services such as OWL-S strive to fill this "semantic gap", but without tools for

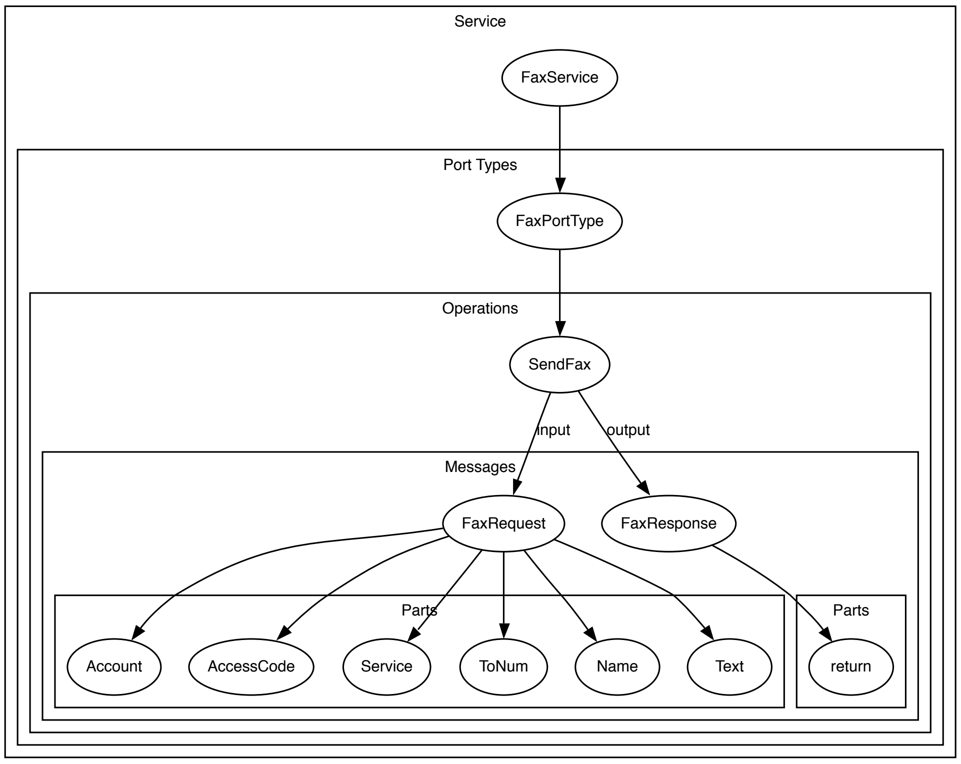


Figure 2.3: *A graphical representation of a simple Web Service*

generating semantic annotation, the need for a considerable amount human intervention is not eliminated, but merely shifted to creating this annotation. As opposed to any semantic description, WSDL is, however, generated automatically from the implementation. Therefore, we use WSDL as the basis from which we try to generate semantic information semi-automatically.

2.2.2 Web Service Description Language (WSDL)

As WSDL is the element of the protocol stack that is most relevant to the work in this thesis, we present some key elements of the description language in this section. These explanations are very brief, but should be sufficient to give the necessary background knowledge that is required to understand the rest of this thesis. For further information about the Web Service Description Language, the reader is referred to other literature, for example the official W3C Note on WSDL (Christensen, Curbera, Meredtih & Weerawarana 2001).

Figure 2.3 shows a graphical representation of a simple Web Service that sends faxes. The same Web Service can be found as a source code example in WSDL in appendix A. The individual components of a Web Service as shown in the graph are explained in the following paragraphs.

Service

A service in WSDL is an aggregate of several *ports*. Theoretically, a service could comprise several ports (and thus operations) with completely unrelated functionality. In practice, however, a service is a consecutive programming interface.

Port Types

A *port* in WSDL denotes an address or communication endpoint, that is bound to a port type. A port type is a collection of *operations*.

Operations

An operation corresponds almost directly to a method or function in a programming language. Input and output of an operation is defined through *messages*.

Messages

Messages, consisting of *parts*, define the signature, i.e. the inputs and outputs, of an operation. An operation can have at most three messages defined: An input message, and output message and a fault message. The fault message is sent instead of the output message, if an error occurs during execution. Fault messages allow a straightforward way for exception handling. Each message consists of one or more *parts*.

Parts

A part in WSDL corresponds to a single parameter in an input, output or fault message. A part can be either of a simple datatype such as integer or string, or of a *complex type*.

Complex Types

A complex type, specified in XML schema, is used to define composite type or structure (compare the "struct" keyword in C/C++). Its elements can be either single data elements or other complex types.

2.3 Definitions

While the applications of our research are Web Service annotation and ontology mapping, both of these tasks can be seen as specialisations of mapping vertices in a graph. In this section, we start by introducing a notation for graph matching. We continue with some definitions that serve as a foundation for explaining the algorithms that we will present in the remainder of this thesis.

2.3.1 Mapping Function

Related Work

The annotation of Web Services with semantic concepts can be seen as a special form of ontology mapping, where a Web Service is treated as a very simple form of an ontology. Therefore, it is straightforward to use the same notation for both problems. Ehrig and Staab in (Ehrig & Staab 2004) define a mapping function for ontologies.

Definition 1 (Ehrig and Staab, (Ehrig & Staab 2004))*.* We define an ontology mapping function, *map*, based on the vocabulary, $\mathcal{E}$, of all terms $e \in \mathcal{E}$ and based on the set of possible ontologies, $\mathcal{O}$, as a partial function

$$\text{map} : \mathcal{E} \times \mathcal{O} \times \mathcal{O} \rightharpoonup \mathcal{E}$$

with $\forall e \in O_1(\exists f \in O_2 : \text{map}(e, O_1, O_2) = f \vee \text{map}(e, O_1, O_2) = \perp)$.

However, this notation has some problems. Clearly, the vocabulary $\mathcal{E}$ in the domain is not the same as the vocabulary $\mathcal{E}$ in the codomain. Ehrig and Staab circumnavigate this issue by introducing a set of ontologies $\mathcal{O}$, where according to the definition every element $O \in \mathcal{O}$ is itself a set of entities, so that each e is both an element of the vocabulary $\mathcal{E}$ and an ontology O and imposing constraints on e and f.

In various approaches such as (Melnik, Molina-Garcia & Rahm 2002) or (Jeh & Widom 2002), the mapping problem is cast as a graph matching problem. For the notation in this thesis, we follow that notion and treat the entities that we are trying to match as nodes in a graph.[4] We use the words "node", "vertex" and "entity" synonymously.

Graph Interpretation

We define the mapping problem as identifying pairs of vertices from two edge-labelled directed graphs.

In the case of the ontology mapping problem, the vertices represent entities in the ontology (i.e. classes and properties). The arcs denote relations between these entities, and the labels signify the kind of relation, e.g. "subclass of", "domain" or "range".[5]

For the Web Service matching problem, vertices in the graph correspond to components of a Web Service, e.g. operations, complex types or message parts. Arcs denote relations such as "message part belongs to operation".

Definition 2*.* Let $G = (V, A)$ and $G' = (V', A')$ be two directed graphs with V and V' as their set of vertices and A and A' as their set of arcs. We define two partial functions

[4]The papers mentioned, (Melnik et al. 2002) and (Jeh & Widom 2002) make use of derived graphs such as the pairwise connectivity graph and the similarity propagation graph in (Melnik et al. 2002) or the node-pairs graph in (Jeh & Widom 2002). We do not use such derived graphs here.

[5]Note that the terms relation, domain and range in an ontological sense should not be confused with the same terms in a mathematical sense.

that are not necessarily surjective and injective as:

$$\text{map} : V \rightharpoonup V'$$

and

$$\text{map}' : V' \rightharpoonup V$$

From non-surjectivity and non-injectivity follows that we do not require that $\text{map}'(\text{map}(v)) = v$.

Although potentially it would also be useful to map edges instead of just vertices, we restrict ourselves to the case where edge labels are drawn from a fixed, common vocabulary. As illustrated above, this is the case in the ontology mapping scenario. We defer the generalised approach that also maps edges to future work.

2.3.2 Similarity Function

Definition 3. To measure the similarity between vertices from G and G', we define similarity functions as:

$$\text{sim} : V \times V' \to [0, 1]$$

and

$$\text{sim}' : V' \times V \to [0, 1]$$

We allow different similarity functions to be used and denote them with an index. We define that $\text{sim}(v, v') = 1$ for perfect similarity and $\text{sim}(v, v') = 0$ if v and v' are completely dissimilar. In general, we do not require that $\text{sim}(v, v') = \text{sim}'(v', v)$, although this assertion might be true for certain sim_k. When we use sim in the remainder of the thesis we will explicitly state if $\text{sim}(v, v') = \text{sim}'(v', v)$ holds or not where this issue is of importance.

We define a binary similarity function sim_{bin} that has a direct correspondence to *map*. Since $\text{map}(v) = v'$ does not imply $\text{map}'(v') = v$, $\text{sim}_{\text{bin}}(v, v') = \text{sim}'_{\text{bin}}(v', v)$ does not hold.

Definition 4. We define sim_{bin}, a similarity measure according to definition 3, as follows:

$$\text{sim}_{\text{bin}}(v, v') = \begin{cases} 1 & \text{if } \text{map}(v) = v' \\ 0 & \text{otherwise} \end{cases}$$

The definition for sim'_{bin} is analogous.

2.3.3 Taxonomies

In the problems that we will consider in the remainder of the thesis the task is not always to map nodes in similarly structured graphs to one another. In the Web Services task for example, the mapping and the similarity between two services are merely means to annotate (the elements in) a service with labels drawn from an arbitrary taxonomy.

In that case, the structural similarity between a service and the taxonomy itself is not meaningful. In fact, the taxonomy might not even have a hierarchical structure, but could be a (flat) list of labels.

While it is important to note the difference between these two problems, we can still use a similar notation for mappings and similarities.

Definition 5. Let $G = (V, A)$ be a directed graph with V as its set of vertices and A as its set of arcs. Let $\mathcal{T}$ be a taxonomy and $t \in \mathcal{T}$ a class label drawn from $\mathcal{T}$.

We then define a partial function:

$$\text{map}_{\text{tax}} : V \rightharpoonup \mathcal{T}$$

Definition 6. To measure the similarity between a vertex from G to a class in $\mathcal{T}$, we define a similarity function as:

$$\text{sim}_{\text{tax}} : V \times \mathcal{T} \to [0, 1]$$

We omit the subscript if it is obvious from the context that we are measuring the similarity between a node in a graph and a label in a taxonomy rather than the similarity between two nodes.

The mappings and similarities between vertices and classes may derive from a machine learning algorithm, see section 2.4 for a brief introduction. As a simple example, consider a nearest neighbour-classification, see also section 2.4.2. The mapping from a vertex v to a class t then derives from a known mapping from a vertex v' to class t and a mapping between v and v' based on the similarity between these two vertices. The class in the taxonomy that is assigned to v is obtained by first mapping v to some v' and then using the known mapping from v' to a class in the taxonomy, i.e. $\text{map}_{\text{tax}}(v) = \text{map}_{\text{tax}}(\mathit{map}(v)) = \text{map}_{\text{tax}}(v') = t$. The mapping $\text{map}_{\text{tax}}(v) = t$ is also known as *training data*.

Definition 7. We define *training data* as a set $\mathcal{E}$ of ordered pairs (v, t) with $v \in V$ and $t \in \mathcal{T}$.

2.3.4 Subsets of V

Depending on the application, it is not always desirable that all $v \in V$ can be mapped to all $v' \in V'$ or $t \in \mathcal{T}$. If we consider ontology mapping, we may want to prohibit that classes are mapped to individuals or properties or vice versa.[6] In the Web Services context, it makes no sense to map operations to parameters. To facilitate for such cases, we define subsets of V resp. V' or $\mathcal{T}$ that denote types of nodes. In the ontology case, we denote $C \subset V$ resp. $C' \subset V'$ as the set of classes and $P \subset V$ resp. $P' \subset V'$ as the set of properties. In the Web Services case, we denote $S \subset V$ as the set of services, $O \subset V$ as the set of operations and $D \subset V$ as the set of parts, complex types and

[6]Although sometimes it might actually be the case that things are represented as classes in one ontology appear as individuals in another, we do not consider such cases in this work.

elements (i.e. the individual parameters of a service). Since in the Web Service case we are mapping the components of a service to a taxonomy rather than another service, we define $S' \subset \mathcal{T}$, $O' \subset \mathcal{T}$ and $D' \subset \mathcal{T}$ as analogous subsets of the taxonomy.

2.3.5 Related Entities

Definition 8. We define a function from the set of vertices to the power set of vertices so that for a given vertex the function finds all adjacent vertices:

$$\mathrm{rel} : V \rightarrow 2^V$$

Let $G = (V, A)$ be a digraph with the set of vertices V and arcs A as a set of ordered pairs of vertices. Then we define:

$$\mathrm{rel}(v) = \{x | v, x \in V \wedge (v, x) \in A\}$$

The definition of $\mathrm{rel}' : V' \rightarrow 2^{V'}$ is analogous.

Definition 9. We define a function from the set of vertices and the set of labels L to the power set of vertices so that for a given vertex the function finds all vertices adjacent through an arc with a given label:

$$\mathrm{rel} : V \times L \rightarrow 2^V$$

Let $G = (V, A)$ be a digraph with the set of vertices V and labelled arcs A as a set of ordered triples $(v, w, l) \in V \times W \times L$. Then we define:

$$\mathrm{rel}(v, l) = \{x | v, x \in V \wedge (v, x, l) \in A\}$$

The definition of $\mathrm{rel}' : V' \times L \rightarrow 2^{V'}$ is analogous. We allow $\mathrm{rel}(v, l)$ to be denoted as $\mathrm{rel}_l(v)$.

2.3.6 Feature Vector Functions

Most of the approaches that we present in this thesis make use of a feature vector representation of entities (i.e. vertices in our graph model, see above). This feature vector representation can be used either to directly compute a similarity value between entities or as an input for a machine learning algorithm. We write $\vec{f}(v)$ respectively $\vec{f}'(v')$ to denote a (general) feature vector representation of an entity. For the definition of specific representation, see below. In the following, we write all feature vector functions as functions $V \rightarrow \mathbb{R}^n$, the definition for V' is analogous. This vector representation of an entity should not be confused with the matrix representation of a graph.

Definition 10. We define two feature vector functions $\vec{f}$ and $\vec{f}'$ that map an entity to a vector representation of that entity.

$$\vec{f} : V \to \mathbb{R}^n$$

$$\vec{f'} : V' \to \mathbb{R}^n$$

Following (Neville & Jensen 2000), we distinguish between *static intrinsic*, *dynamic intrinsic*, *static extrinsic* and *dynamic extrinsic* features.

Static Intrinsic Features

Static intrinsic features are attributes that are inherent to an entity. Let us consider the example of link-based web page classification. The task is to classify a web page into one of several categories. We can use information from the web page that is to be classified as well as from web pages connected with links. In this example, static intrinsic features of a web page are the words in the text of the page.

More formally, the static intrinsic feature vector depends on a single entity itself and on the mode of representation that we choose. Since the entities that we want to map are characterised through text associated with them (e.g. labels, comments or instance data), it is straightforward to resort to the well-known vector space model from information retrieval (e.g. (van Rijsbergen 1979) or (Salton & McGill 1983)). In the term vector model, also referred to as "bag of words" model, a document is represented by the set of its words. This set is usually cast to a vector by defining an arbitrary order on the set of words and assigning each word in the vocabulary to an element in the vector.

We can choose between different options for defining the elements of the term vector. The elements of the vector can be binary (a term is either present or absent), integer numbers (word counts) or real numbers (weighted word counts, for example the well known *TFIDF*, e.g. (Salton & McGill 1983)). In our definition, we assume $\mathbb{R}^d$ as the codomain, although depending on the representation that we choose the range might be only $[0,1]^d$ or even $\{0,1\}^d$. The dimensionality d of the vector is a constant and depends on a lexicon of known terms.

Definition 11. We define a static intrinsic feature vector function as a function of an entity:

$$\vec{\mathrm{si}} : V \to \mathbb{R}^d$$

For $\vec{\mathrm{si}}$ and all subsequently defined feature vector functions, we allow different implementations to coexist. We denote different implementations with subscripts, e.g. we write $\vec{\mathrm{si}}_k(v)$.

Dynamic Intrinsic Features

Dynamic intrinsic features are also inherent to an entity, but they are dynamic in the sense that their value can change as we get more information about that entity. In the web page example (see above), this corresponds to the prediction of a web page's class. This prediction might change as we get more information, e.g. if we get to know the class assigned to linked pages, this might change our prediction for the current page.

According to our more formal definition, dynamic intrinsic features of an entity v are its mapping $\mathrm{map}(v)$ and the similarities $\mathrm{sim}(v, v')$ for all $v' \in V'$. Note that the dynamic intrinsic features are typically what we want to compute. In particular, this means that the dynamic intrinsic features are initially unknown. If the mapping of an entity is known a priori, we call that entity *training data*.

To align the dynamic intrinsic with the static intrinsic features, we define a vector function for dynamic intrinsic features.

Definition 12. We define a dynamic intrinsic feature vector function as a function of an entity:

$$\vec{\mathrm{di}} : V \to \mathbb{R}^{|V'|}$$

Analogous to the matrix representation of a graph, we impose an arbitrary total order on V' and denote the first element of V' as v'_0 and the subsequent elements as v'_n for all $n < |V'|$. Then we define $\vec{\mathrm{di}}$ as follows:

$$\vec{\mathrm{di}}(v) = [\mathrm{sim}(v, v'_0), \mathrm{sim}(v, v'_1), \ldots, \mathrm{sim}(v, v'_{|V'|-1})]$$

Static Extrinsic Features

Extrinsic features derive from the relationship of an entity to other entities. Static extrinsic features are static intrinsic features of related entities. In the web page example, this is text that comes from a related web page. We have assume that neither the text of a web page nor its links change while we classify it, so the text from linked web pages also does not change and remains static.

Definition 13. We define a static extrinsic feature vector function as a function of an entity.

$$\vec{\mathrm{se}} : V \to \mathbb{R}^d$$

Assuming a commutative and associative operator $\oplus$ on $\mathbb{R}^d$ and a function rel as per definition 8, we define $\vec{\mathrm{se}}(v)$ as some combination $\oplus$ of the intrinsic features $\vec{\mathrm{si}}(x)$ (see definition 11) of all related entities $x \in \mathrm{rel}(v)$.

$$\vec{\mathrm{se}}(v) = \bigoplus_{x \in \mathrm{rel}(v)} \vec{\mathrm{si}}(x)$$

Dynamic Extrinsic Features

Dynamic extrinsic features are dynamic intrinsic features of related entities. In the web page example, dynamic extrinsic features come from the class prediction that we make for linked web pages. These predictions may change over time, as we make more accurate predictions. Note that dynamic extrinsic features are dynamic intrinsic features of linked pages.

Like the dynamic intrinsic features, the dynamic extrinsic features are initially unknown. The dynamic extrinsic features are based on the similarities of related entities. This means that the dynamic extrinsic features are unknown until these similarities have been computed.

Definition 14. We define a dynamic extrinsic feature vector function as a function of an entity.

$$\vec{\mathrm{de}} : V \to \mathbb{R}^{|V'|}$$

Assuming a commutative and associative operator $\oplus$ on $\mathbb{R}^d$ and a function rel as per definition 8, we define $\vec{\mathrm{de}}(v)$ as some combination $\oplus$ of the dynamic intrinsic features $\vec{\mathrm{di}}(x)$ (see definition 12) of all related entities $x \in \mathrm{rel}(v)$.

$$\vec{\mathrm{de}}(v) = \bigotimes_{x \in \mathrm{rel}(v)} \vec{\mathrm{di}}(x)$$

2.3.7 Similarity Between Vectors

To compute the similarity between two entities based on their feature vector representation, we resort to some well known (e.g. (van Rijsbergen 1979) or (Salton 1989)) metrics from information retrieval.

Inner Product

The simplest similarity measure for two vectors is the inner product (or dot product). For two binary term vectors, the inner product is the number of terms that the two entities have in common, i.e. the intersection of the two term sets.

$$\mathrm{sim}_{\mathrm{dot}}(v, v') \overset{\mathrm{def}}{=} \vec{f}(v) \cdot \vec{f}'(v')$$

Failure to normalise the similarity measure can lead to counterintuitive results (see (van Rijsbergen 1979)). For entities with long texts or, in the case of an extrinsic feature vector, for entities with many relations, the intersection between two vectors will usually be larger, thus overweighting the importance of these entities. There exist several possibilities to normalise this similarity measure.

Cosine Coefficient

A straightforward normalisation that has a geometric interpretation is the cosine coefficient. The value is equivalent to the cosine of the angle between the two vectors $\vec{f}(v)$ and $\vec{f}'(v')$.

$$\mathrm{sim}_{\cos}(v, v') \overset{\mathrm{def}}{=} \frac{\vec{f}(v) \cdot \vec{f}'(v')}{\sqrt{(\vec{f}(v))^2 (\vec{f}'(v'))^2}}$$

Dice Coefficient

The Dice coefficient normalises the dot product of the vectors using their average length.

$$\text{sim}_{\text{dice}}(v, v') \stackrel{\text{def}}{=} 2\frac{\vec{f}(v) \cdot \vec{f'}(v')}{(\vec{f}(v))^2 + (\vec{f'}(v'))^2}$$

Jaccard Coefficient

The Jaccard coefficient is similar to the Dice coefficient. The intersection of the two vectors is normalised by their union.

$$\text{sim}_{\text{jaccard}}(v, v') \stackrel{\text{def}}{=} \frac{\vec{f}(v) \cdot \vec{f'}(v')}{(\vec{f}(v))^2 + (\vec{f'}(v'))^2 - \vec{f}(v) \cdot \vec{f'}(v')}$$

Overlap Coefficient

The overlap coefficient normalises the dot product by the length of the shorter of the two vectors.

$$\text{sim}_{\text{overlap}}(v, v') \stackrel{\text{def}}{=} 2\frac{\vec{f}(v) \cdot \vec{f'}(v')}{\min((\vec{f}(v))^2, (\vec{f'}(v'))^2)}$$

2.4 Introduction to Machine Learning

This section gives a very brief overview of some Machine Learning techniques with the intention to make the rest of the thesis intelligible to people who are not experts in Machine Learning. It is by no means a complete overview. For more detailed information, the reader is referred to other literature, for example Mitchell's well known textbook on Machine Learning (Mitchell 1997).

In this overview, we start with a definition of Machine Learning and briefly present some basic techniques that will be used in the remainder of the thesis. Mitchell defines that a computer program learns, if it improves in performing its task with experience.

Definition 15 (Mitchell, (Mitchell 1997)). 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, measured by P, improves with experience E.

A common way of casting this definition is the classification problem, the task to attach a label drawn from a set of *class labels* to an entity. An algorithm is trained to map a *feature vector* to a class label. This vector is a series of observations about the entity that is to be classified. In the common text classification task, this is usually a term vector (see section 2.3.6). In this scenario, the task T is to classify documents into categories and experience E consists of a set of already labelled documents referred to as *training data*. In general Machine Learning tasks, documents are usually referred to as instances. For the performance measure P, we can use the well-known *precision* and *recall* metrics.

2.4.1 Accuracy, Precision, Recall and F1

Accuracy, Precision, Recall and F1 are well-known metrics for evaluating classifiers. While Accuracy is simply the percentage of correctly classified instances, Precision, Recall and F1 measure the performance of a classifier on a per-class basis. Precision measures the correctness of the results for a given class and Recall measures the completeness. The F1-Measure is the harmonic mean of Precision and Recall.

Definition 16. The *Accuracy* of a classifier is the fraction of instances correctly classified by the classifier over all instances classified by the classifier.

Definition 17. The *Precision* Prec(C) for a class C is the fraction of instances *correctly* labelled as C over the total number of instances that were labelled as C.

Definition 18. The *Recall* Rec(C) for a class C is the fraction of instances correctly labelled as C over the total number of instances that actually belong to C.

Definition 19. The *F1* measure F1(C) for a class C is the harmonic mean of the Precision and Recall for C.

$$\mathrm{F1(C)} = \frac{2\mathrm{Prec(C)Rec(C)}}{\mathrm{Prec(C)} + \mathrm{Rec(C)}}$$

Consider a simple binary classification problem with two classes A and B. When we classify a number of instances, we can construct a *confusion matrix* (also referred to as *contingency table*) as in figure 2.4, that lists counters for each predicted and for each actual class label. In this example, $a + b$ is the number of instances that belong to class A, and $c + d$ is the number of instances classified as class B. Given definitions 16–19 we can now easily calculate Accuracy as well as Precision and Recall for A and B. (The value for F1(A) and F1(B) follows directly from definition 19.)

$$\mathrm{Accuracy} = \frac{a+d}{a+b+c+d}$$

$$\mathrm{Prec(A)} = \frac{a}{a+c}$$

$$\mathrm{Prec(B)} = \frac{d}{b+d}$$

$$\mathrm{Rec(A)} = \frac{a}{a+b}$$

$$\mathrm{Rec(B)} = \frac{d}{c+d}$$

To obtain a Precision or Recall metric that characterises the behaviour of a classifier over all classes, there are two obvious ways to average the per-class measures that are known as *microaveraging* and *macroaveraging*. Microaveraging considers all classes

predicted class		A	B
actual class	A	a	b
	B	c	d

Figure 2.4: *An example for a* confusion matrix *or* contingency table

as one and then computes Precision and Recall where as macroaveraging computes Precision and Recall separately for each class and then averages. Note that if each instance is labelled with exactly one label microaveraged Precision and Recall equals Accuracy. Macroaveraged Precision, Recall or F1 is of particular interest if the class distribution is skewed because it treats all classes as equally important, as opposed to Accuracy, which is dominated by large classes. For a discussion of these evaluation metrics in greater detail, see (Lewis 1991).

2.4.2 Nearest Neighbour

If we have as similarity measure defined over entities that we want to classify, it is straightforward to define a classifier that makes a prediction based on the known label of instances in the training set that are similar to the instance in question. As a similarity measure for instances we can resort to the measures introduced in section 2.3.7, but for tasks other than text classification it can make sense to use Euclidian distance instead. In its simplest form, a *1-Nearest Neighbour* classifier simply outputs the label of the single nearest instance to the instance in question. Distance-weighted classifiers that incorporate the labels of the *k-Nearest Neighbours* can be more robust against noise in the training data.

2.4.3 Naive Bayes

The well-known Naive Bayes classifier is based on the calculation of conditional probabilities of the instance belonging to a certain class given the observation given in form of the feature vector, i.e. let F_i denote random variables based on the feature vector, then we are interested in computing $P(C|F_1, \ldots, F_n)$ for each class C. Using Bayes' theorem and making the "naive" assumption that all features are conditionally independent given the class variable C this is equivalent to computing $P(C)\prod_{i=1}^{n} P(F_i|C)$. Note that the parameters of this term can be easily estimated from frequency counts obtained from the training data. The Naive Bayes classifier works reasonably well for most datasets, although the independence assumption is often violated in practice.

2.4.4 Support Vector Machine

The Support Vector Machine or SVM is a classification method for binary classification tasks that is based on computing a separating hyperplane that maximises the margin between examples of both classes. The original Support Vector Machine is a linear classifier, however, with the "kernel trick" it is possible to extend the SVM to non-linear classification problems by using a a non-linear kernel function. The SVM classifier is based on ideas by V. Vapnik (Vapnik 1979, Vapnik 1982). For a more detailed description on SVM, the reader is referred to Bernhard Schölkopf's PhD thesis (Schölkopf 1997). A fast way to compute an SVM classifier is the Sequential Minimal Optimization algorithm (SMO) by John Platt (Platt 1999). SVMs for text classification have been extensively studied by Thorsten Joachims, e.g. (Joachims 1999).

2.4.5 Ensemble Methods

Ensemble techniques that combine the predictions of several base classifiers have been demonstrated to be an effective way to reduce the error of a base learner across a wide variety of tasks. The basic idea is to vote together the predictions of a set of classifiers that have a different view on the data.

A view is a subset of all features that is independent from the rest of the features and should be sufficient for a classifier to make a prediction. Because the features in different views are independent from each other, the hope is that the prediction errors of classifiers based on these views are independent as well. Ideally, the classifiers that make the wrong prediction on any given instance are outvoted.

If there are no multiple views on the data, the same effect can be achieved by using multiple different algorithms or by training the same algorithm (base classifier) in different ways, i.e. by modifying its training set. There is a strong body of theory explaining why ensemble techniques work, e.g. (Dietterich 2000).

Bagging

Bagging or *Bootstrap Aggregating* (Breiman 1996) is an ensemble method that trains multiple classifiers on a different random subset of the original training set. Algorithm 1 illustrates Bagging for binary classification tasks with the class label t being either $+1$ or -1. The training phase of the algorithm for multiple classes is analogous, on classification phase majority voting is used. h denotes a single hypothesis learned by the base algorithm.

Algorithm 1 Bagging

```
/* To train on $\mathcal{E} = \{\dots, (v_i, t_i), \dots\}$ ($t_i = \pm 1$) */
for $l = 1, 2, \dots, K$ do
    Let $\mathcal{E}_l$ be a set of $N$ elements from $\mathcal{E}$ drawn at random with replacement
    $h_l$ = Learn from $\mathcal{E}_l$
end for
/* To classify instance $x$ */
return sign$\left[\sum_{l=1}^{K} h_l(x)\right]$ /* Majority Voting */
```

Boosting

Boosting creates multiple classifiers by emphasising instances that were misclassified initially. Shapire's original boosting algorithm (Shapire 1990) uses only three classifiers. The later version AdaBoost (Freund & Shapire 1997) uses multiple rounds, instance weights and weighted voting to combine the results of the multiple classifiers. Boosting (and the difference between AdaBoost and our own Triskel algorithm) is explained later in algorithm 10 in section 7.3.2.

Co-Training

Co-Training (Blum & Mitchell 1998) is an ensemble technique that uses unlabelled data to improve classification accuracy, but requires multiple views to be present. The original Co-Training algorithm is presented in algorithm 2.

Algorithm 2 Co-Training

```
/* To train from labelled instances with two views $\mathcal{L} = \{\ldots,(v_i^1, v_i^2, t_i),\ldots\}$ and unlabelled instances $\mathcal{U}$ with two views. */
Let $\mathcal{U}'$ be $u$ examples from $\mathcal{U}$ chosen at random
for $l = 1, 2, \ldots, K$ do
    Train $h_1$ on the set $\{\ldots,(v_i^1, t_i),\ldots\}$
    Train $h_2$ on the set $\{\ldots,(v_i^2, t_i),\ldots\}$
    Let $h_1$ label $p$ positive and $n$ negative examples from $\mathcal{U}'$
    Let $h_2$ label $p$ positive and $n$ negative examples from $\mathcal{U}'$
    Add these newly labelled examples to $\mathcal{L}$ and remove them from $\mathcal{U}'$
    Choose $2p + 2n$ examples from $\mathcal{U}$ at random and add them to $\mathcal{U}'$
end for
Train $h_1$ on the set $\{\ldots,(v_i^1, t_i),\ldots\}$
Train $h_2$ on the set $\{\ldots,(v_i^2, t_i),\ldots\}$
/* To classify instance $x$ */
return $h_1(x) \times h_2(x)$
```

2.4.6 Unsupervised Clustering

The Machine Learning methods presented so far all have in common that they assume the presence of at least some labelled training instances. However, given some similarity measure, it is still possible to identify instances that should be grouped together.

Clustering algorithms can be divided into hierarchical and partitional algorithms. Hierarchical algorithms can be further divided into top-down (divisive) and bottom-up (agglomerative) methods.

Hierarchical Agglomerative Clustering

The very common hierarchical agglomerative clustering (or HAC-clustering) is quite straightforward. It is presented in a very generic form in algorithm 3. Variants of

Algorithm 3 Hierarchical Agglomerative Clustering

```
Place each instance in its own cluster
Compute pairwise similarities between all clusters
repeat
  Merge the two most similar clusters
  Compute similarities of all clusters to the newly created cluster
until termination criterion is met
```

HAC-clustering differ in the termination criterion and in the way the similarity between clusters is computed.

The most common way to define the similarity between two clusters is use the similarity of their *centroids*. The centroid of a cluster is an imaginary instance in the centre of a cluster. If the instances are represented as vectors, the centroid is the arithmetic mean of all instance vectors in the cluster. Thus, the similarity for two clusters of vectors is:

$$\text{sim}(A, B) = \text{sim}(\frac{\sum_{a \in A} a}{|A|}, \frac{\sum_{b \in B} b}{|B|})$$

In *single link* clustering, the similarity between two clusters A and B is defined as the greatest pairwise similarity between instances from A and B:

$$\text{sim}(A, B) = \max_{a \in A, b \in B} \text{sim}(a, b)$$

Single link clustering has the disadvantage of producing chain-like clusters, since it is sufficient for two clusters to be merged, if they are very similar at a single point.

In *complete link* clustering, the similarity between two clusters A and B is defined as the smallest pairwise similarity between instances from A and B:

$$\text{sim}(A, B) = \min_{a \in A, b \in B} \text{sim}(a, b)$$

Complete link clustering produces clusters that are coherent.

k-Nearest Neighbour and k-Means Clustering

The simplest partitional clusterer is the k-Nearest Neighbour[7] clusterer. It works by selecting k instances as "seeds" for clusters and then subsequently assigning instances to the cluster with the nearest seed. The refinement k-Means then calculates the centroid of these clusters and uses these as seeds for the next iteration. This method is outlined in algorithm 4. Basically, k-Nearest Neighbour is a k-Means clusterer with just one iteration.

Because of the requirement to calculate pairwise similarities, the complexity for single-link clustering is $O(n^2)$, and the complexity for complete-link clustering is $O(n^2$

[7]Although somewhat similar, the unsupervised k-Nearest Neighbour clusterer should not be confused with the k-Nearest Neighbour algorithm for supervised learning!

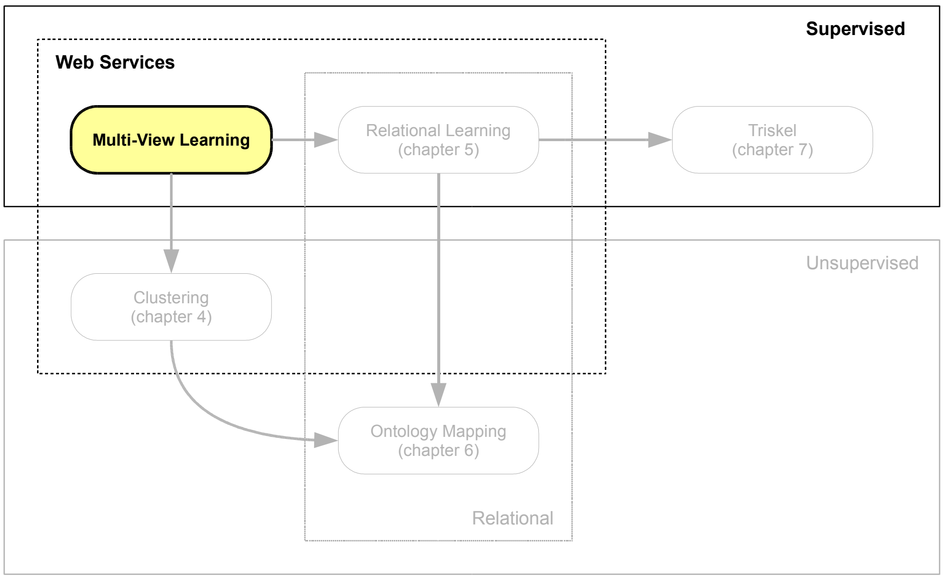
Algorithm 4 The k-Means and k-Nearest Neighbour Clustering Algorithms

```
Select a set of $k$ instances $S = \{c_1, \ldots, c_k\}$
Create a cluster $C_i$ for each $c_i \in S$
repeat
  for each instance $x$ do
    Compute similarity between $x$ and all instances $c_i \in S$
    Assign $x$ to the cluster $C_i$ with the most similar $c_i$
  end for
  /* If k-Nearest Neighbour, stop here. */
  for each cluster $C$ do
    Replace $c_i$ with the centroid of $C_i$
  end for
until convergence
```

log n). Partitional clustering algorithms such as the k-Nearest Neighbour clusterer are linear in the number of instances. The drawback is, however, that the number of clusters k must be specified in advance. Cutting et al. in (Cutting, Pedersen, Karger & Tukey 1992) propose the "Buckshot" and "Fractionation" algorithms to find an intial set of k instances for a partitional clusterer in an effective way.

Chapter 3

Categorising Web Services



In this chapter, we discuss the problem of classifying a web wervice into a category and treating this as a text classification problem. We show that an approach that splits the available evidence and uses an ensemble of several classifiers works better than a simple classifier. While classifying a web service as a whole may be useful for example to automatically place it in a taxonomy for use within a UDDI registry, it is only the first step towards fully annotated web services. The experimental results obtained with the setup that is described in this chapter was an encouragement to proceed towards annotating the components of a web service as well.

3.1 Problem Formulation

Following our definitions from section 2.3, we assume a web service as a directed graph $G = (V, A)$. We assume a taxonomy $\mathcal{T} = \{t_1, t_2, \ldots\}$ of Web Service categories.

We treat the determination of a web service's category as a text classification problem, where the text comes from the web service's WSDL description. Unlike standard texts, WSDL descriptions are highly structured. Our experiments demonstrate that selecting the right set of features from this structured text improves the performance of a learning classifier. By combining different classifiers it is possible to improve the performance even further, although for both the feature selection and the combination no general rule exists. The splits of the texts were treated as given from the structure of a web service description and is in the following section discussed in greater detail. Note that the problem of finding an appropriate split is different from automatic feature selection in general machine learning problems.

Formally, the task of classifying the category of a service can be defined as finding a mapping $\text{map}_{\text{tax}}(v)$ with v root node of G, given some training data $\mathcal{E}$ as per definition 7. In this task, v is always the root node, because we are only interested in classifying the service as a whole at this time, and not any inner components of the web service. Note that we only classify one node in each graph. We have neither mappings nor even a taxonomy for other nodes but the root. All mappings $\text{map}_{\text{tax}}(x)$ with x being any other node but the root node are undefined. This means that we cannot use dynamic extrinsic features to determine $\text{map}_{\text{tax}}(v)$. We can only use *static intrinsic features* $\vec{\text{si}}(v)$ and *static extrinsic features* $\vec{\text{se}}(v)$.

Note that we always assume that a web service belongs to one category only. This is a reasonable simplification, if we assume that a new class is introduced where services would otherwise be in two categories. For example, services that would belong to two categories like "books" and "e-commerce" are instead assigned to a category "book-selling". If, however, classifications are desired where services should be placed into more than one class, other learning algorithms respectively meta-classification schemes are required. This is, however, not discussed in this thesis.

In the following sections, we describe our web service corpus, describe the methods we used for classification, and evaluate our approach.

Category taxonomy $\mathcal{T}$ and number of web services for each category			
Business (22)	Communication (44)	Converter (43)	Country Info (62)
Developers (34)	Finder (44)	Games (9)	Mathematics (10)
Money (54)	News (30)	Web (39)	*discarded* (33)

Figure 3.1: *Web service categories in $\mathcal{T}$.*

3.2 Web Services Corpus

We gathered a corpus of 424 web services from SALCentral.org, a web service index. These web services were then manually classified into a taxonomy $\mathcal{T}$. To avoid bias, the person was a research student with no previous experience with web services. The person had the same information as given on SALCentral.org, and was allowed to inspect the WSDL description if necessary. The person was advised to adaptively create new categories while classifying the web services and was allowed to arrange the categories as a hierarchy.

The 424 web services were classified by our assistant into 25 top level categories. The actual taxonomy that was created consisted of more classes organised in a hierarchy, but in order to simplify the task for this first experiment we decided to use only the classes from the taxonomy that were direct descendants of the root.

We then discarded categories with less than seven instances, leaving 391 web services in eleven categories that were used in our experiments. The discarded web services tended to be quite obscure, such as a search tool for a music teacher in an area specified by ZIP code. Even for a human classifier, these services would be extremely hard to classify. Note that the distribution after discarding these classes is still highly skewed, ranging from nine web services in the "Games" category, to 62 services in the "Country Information" category. The list of categories used eventually is shown in figure 3.1.

3.3 Ensemble Learning

As shown in Fig. 3.2, the information available to our categorisation algorithms comes from two sources. First, the algorithms use the web service description in the WSDL format, which is always available to determine a service's category. Second, in some cases, additional descriptive text is available, such as from a UDDI entry. In our experiments, we use the descriptive text provided by SALCentral.org, since UDDI entries were not available. We parse the port types, operations and messages from the WSDL and extract names as well as comments from various "documentation" tags. We do not extract standard XML Schema data types like string or integer, or informations about the service provider. The extracted terms are stemmed with Porter's algorithm (Porter 1980), and a stop-word list is used to discard low-information terms. We assume the vector space model for our text classification task.

We experimented with four bags of words, denoted by A–D. The composition of these bags of words is marked in Fig. 3.2. We also used combinations of these bags of

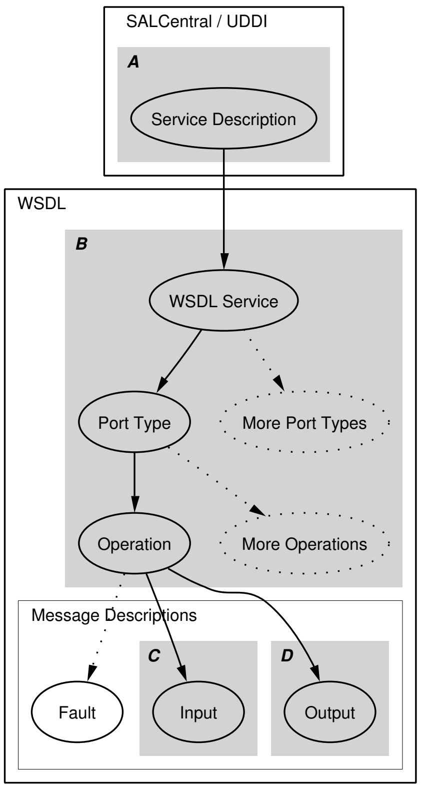


Figure 3.2: *Text structure for our web service corpus*

words, where e.g. *C+D* denotes a bag of words that consists of the descriptions of the input and output messages.

Formally, we define *A* as a short-hand notation for the intrinsic feature vector $\vec{\mathrm{si}}(v)$ as obtained by function $\vec{\mathrm{si}}$ from definition 11. Analogously, we define *B*, *C*, *D* and also *C+D* etc. as short-hand notations for different extrinsic feature vector functions $\vec{\mathrm{se}}_k$ as per definition 13.

We used word frequency counts as elements of the term vectors. In preliminary experiments, we also evaluated the use of *TFIDF* weights, but we could not achieve an improvement. As learning algorithms, we used the Naive Bayes (see section 2.4.3), SVM (see section 2.4.4) and HyperPipes algorithms as implemented in the well-known Weka library (Witten & Frank 1999). To combine two or more classifiers, we multiplied the confidence values obtained from the multi-class classifier implementation.

Formally, let L be the set of subsets of $\{A, B, C, D\}$, that describes the combination of feature vectors that we use: $L \subset 2^{\{A,B,C,D\}}, L \neq \emptyset$. Then:

$$\mathrm{sim}_L = \prod_{s \in L} \mathrm{sim}_s \quad \text{with} \quad \mathrm{sim}_s = \mathrm{sim}_{\sum_{x \in s} x}$$

i.e. sim_s is an implementation of function $\mathrm{sim}_{\mathrm{tax}}$ from definition 6 that uses feature vector $\sum_{x \in s} x$.

We denote a combination of different algorithms or different feature sets by slashes, e.g. Naive Bayes(*A*/*B*+*C*+*D*) denoting two Naive Bayes classifiers, one trained on the plain text description only and one trained one all terms extracted from the WSDL.

We split our tests into two groups. First, we tried to find the best split of bags of words using the terms drawn from the WSDL only (bags of words *B*–*D*). These experiments are of particular interest, because the WSDL is usually automatically generated (except for the occasional comment tags), and the terms that can be extracted from that are basically operation and parameter names. Note that we did not use any transmitted data, but only the parameter descriptions and the XML schema. Second, we look how the performance improves, if we include the plain text description (bag of words *A*).

3.4 Evaluation

We evaluated the different approaches using a leave-one-out methodology: Each web service is classified using a classifier that is trained on all other services. We tested various configurations of algorithms, in figs. 3.3 and 3.4 we show some illustrative examples as well as the setups that performed best.

We used a one-vs-all classification scheme (as implemented in the Weka MultiClass-Classifier) for all classifiers: A separate, binary classifier is trained for each class, and the overall distribution is based on voting. Since SVMs can only handle binary classification problems, this meta-scheme was necessary. In this experiment, even for the Naive Bayes classifier, which is inherently capable of handling multi-class problems, the results were better, if a one-vs-all scheme was used rather than a single classifier. A one-vs-one scheme was considered but not used due to the much higher computational cost.

In a machine learning setting with a split feature set it is also possible to use Co-Training (Blum & Mitchell 1998) to improve classification accuracy, if unlabelled data is present. In preliminary experiments we added 370 unlabelled web services. However, we could gain no advantage using Co-Training.

For a semi-automatic assignment of the category to a web service, it is not always necessary that the algorithm predicts the category exactly, although this is of course desirable. A human developer would also save a considerable amount of work if he or she only had to choose between a small number of categories. For this reason, we also report the accuracy when we allow near misses. Figures 3.3 and 3.4 show how the classifiers improve when we increase this tolerance threshold. For our best classifier, the correct class is in the top 3 predictions 82% of the time.

3.5 Conclusion

Our results show that using a classifier with one big bag of words that contains everything (i.e. *A*+*B*+*C*+*D* for WSDL and descriptions, or *B*+*C*+*D* for the WSDL-only tests) generally performs worst. We included these classifiers in figures 3.3 and 3.4 as baselines. Ensemble approaches where the bags of words are split generally perform

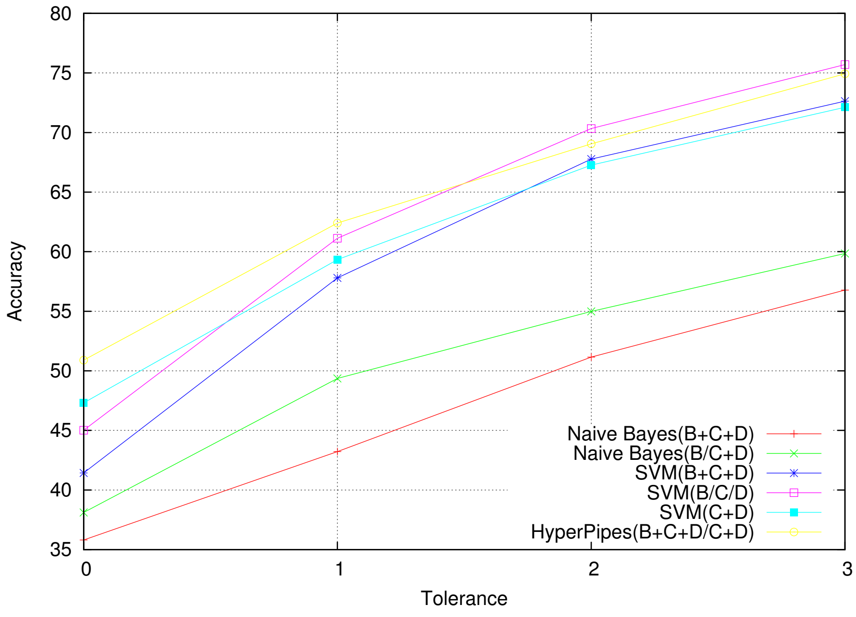


Figure 3.3: *Classification accuracy obtained when using the WSDL only*

	one classifier	two classifiers
Naive Bayes	37.6	53.45 ◇
SVM	47.06	56.78 ◇

Table 3.1: *Percentage of correct predictions made by Naive Bayes and SVM classifiers for a setup with only one classifier vs. a setup where the bag of words was split for two classifiers. Both WSDL and plain text descriptions were used. The improvement due to the split is statistically significant at the* 0.05*-level tested with a Student t-test.*

better. This is intuitive, because we can assume a certain degree of independence between, for example, the terms that occur in the plain text descriptions and the terms that occur in the WSDL description. Table 3.1 shows how the performance improves if we use two classifiers trained on a separate bag of words each instead of one classifier trained on all words. Note that the table shows the results for exact predictions only. The results are statistically significant at the 0.05 confidence level when tested with a Student t-test.[1] What is a bit more surprising is that for some settings we achieve very good results if we use only a subset of the available features, i.e. only one of the bags of words.

In our experiments, the SVM classifier generally performed better than Naive Bayes, except for the setup where we used the plain text descriptions only. Table 3.2 shows a direct comparison of different setups of the Naive Bayes and SVM classifiers on a dataset where both data from WSDL and plain text descriptions were used. Again, the accuracy that is given in the table is for exact predictions.

[1] "Student" is the pen name of William Sealy Gosset (June 13, 1876 – October 16, 1937) who, interestingly, developed the t-test while working as a statistician for the Arthur Guinness & Son brewery in Dublin. The goal behind his work was to enable his employer to monitor the quality of their beer.

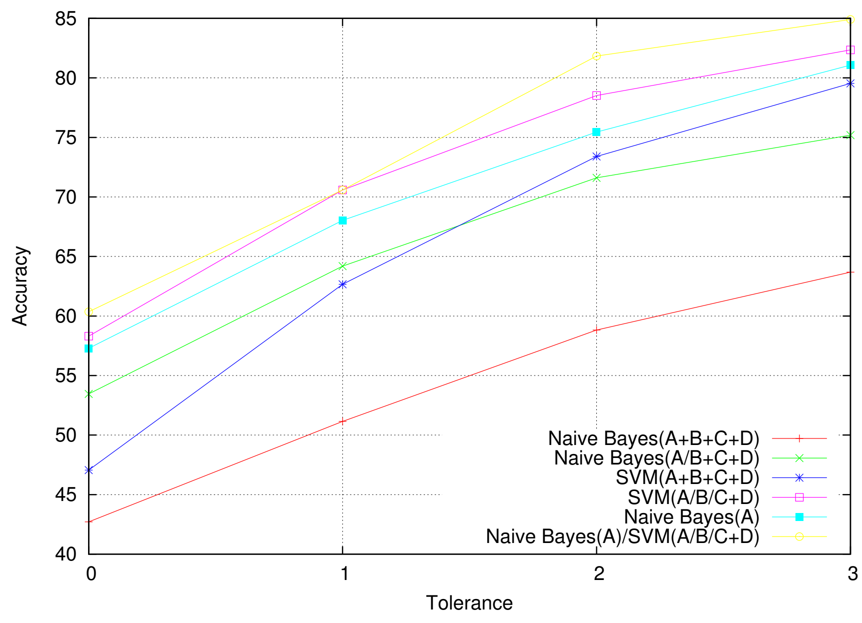


Figure 3.4: *Classification accuracy when using both WSDL and descriptions*

Splits	Naive Bayes	SVM	
1	37.6	47.06	$\diamond$
2	53.45	56.78	
3	49.62	58.31	$\diamond$
4	43.99	54.99	$\diamond$
desc.	55.50	55.75	

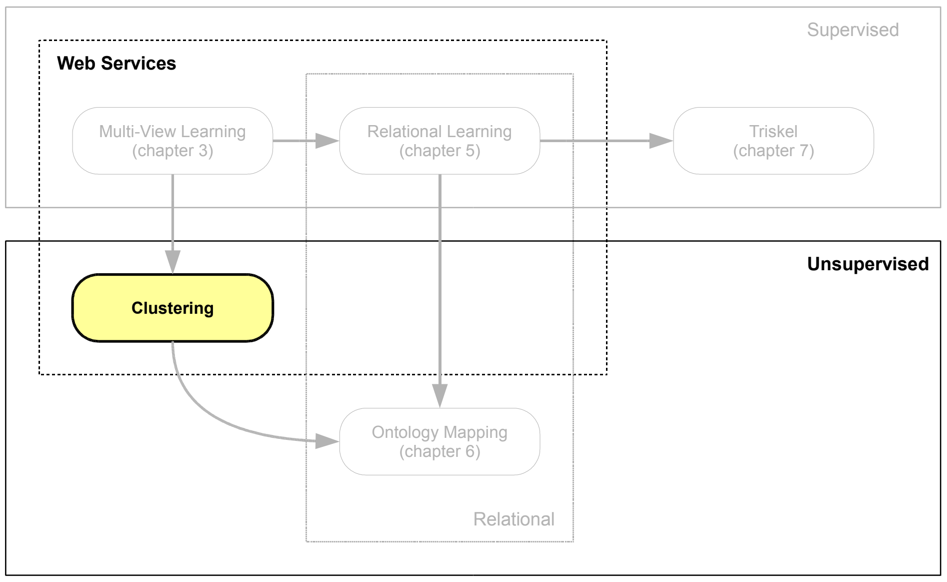
Table 3.2: *Percentage of correct predictions made by Naive Bayes vs. SVM classifiers using both WSDL and plain text descriptions with different number of splits for the ensemble. A* $\diamond$ *denotes results that are significantly better at the* 0.05 *confidence level tested with a Student t-test. "desc" denotes a setup with one classifier that is trained on the plain text descriptions only.*

An ensemble consisting of three SVM classifiers performs well for both the WSDL-only setting and also when including the descriptions. The best results were achieved by other combinations, but we could not find a generic rule for how to best split the available bags of words, as this seems to be strongly dependent on the algorithm and the actual data set. In future work, methods to automatically determine a split that performs well could be developed. This idea as well as the overall results we gained on our web service dataset might be useful for other text classification problems as well, if there is a natural split in different text sources. In an email classification problem, for example, it should be possible to use separate classifier for the subject lines and for the mail body. However, because of the focus of this work on web services, we did not conduct such experiments and leave them to future work.

Today, students in Dublin still monitor the quality of Guinness.

Chapter 4

Clustering Web Services


Supervised
Web Services
Multi-View Learning (chapter 3)
Relational Learning (chapter 5)
Triskel (chapter 7)
Unsupervised
Clustering
Ontology Mapping (chapter 6)
Relational

Since the experiments with supervised Web Service classification yielded encouraging results, the logical next step towards automation is to try a completely unsupervised approach. We introduce a precision/recall-quality metric for clustering that is similar to the use of precision and recall in supervised learning or one-to-one mapping tasks (see also chapter 6) that correlates with some existing metrics and has a probabilistic interpretation. We use a variation of a HAC clustering algorithm that we call "Common Term" that generates the centroid by overlap rather than by averaging and compare it against four other clustering algorithms found in literature.

4.1 Clustering Algorithms

We tested five clustering algorithms on our collection of Web Services. First, we tried a simple k-nearest-neighbour algorithm. Hierarchical centroid based and complete link algorithms (e.g. (van Rijsbergen 1979, Salton & McGill 1983)) serve as representatives of traditional approaches. We also tried a variant of the centroid based clusterer that we call Common-Term, and the Word-IC algorithm (Zamir, Etzioni, Madani & Karp 1997). The Word-IC algorithm, unlike the other clustering algorithms, does not rely on a traditional cosine based similarity measure between the documents (i.e. in the web services in our case), but hierarchically merges clusters based on the number of intersecting words and a global quality function. The global quality function also serves as a halting criterion. The Common Term algorithm halts, when the top level clusters do not share any terms. For the centroid based and complete link algorithms, we used a minimum similarity between documents as a halting criterion. As a baseline, we partition the Web Services into eleven random clusters.

Our Common-Term algorithm differs from the standard centroid based clustering in the way the centroid document vector is computed. Instead of using all terms from all the sub-clusters, only the terms that occur in all sub-clusters form the centroid are used. Like the Word-IC algorithm, our hope is that this leads to short and concise cluster labels.

For our clustering experiments, we used the cosine coefficient for the similarity measure and a TFIDF weighting scheme for the feature vectors.

4.2 Quality Metrics for Clustering

Evaluating clustering algorithms is a task that is considerably harder than evaluating classifications, because we cannot always assign a cluster to a certain reference class. Several quality measures have been proposed; see (Strehl 2002) for a recent survey.

We have evaluated our clustering algorithms using Zamir's quality function (Zamir et al. 1997), and the normalised mutual information quality metric described in (Strehl 2002). We also introduce a novel measure inspired by the well-known precision and recall metrics that correlates well with other quality measures and has a simple probabilistic interpretation.

In the literature on evaluating clustering algorithms, precision and recall have only been used on a per-class basis. This assumes that a mapping between clusters and reference classes exists. The fraction of documents in a cluster that belong to the "dominant" class, i.e. the precision assuming the cluster corresponds to the dominant class, is known as purity. Usage of the purity measure is problematic, if the cluster contains an (approximately) equal number of objects from two or more classes. This clustering might not even be unintuitive, if it is merely the case that the granularity of the clustering is coarser than that of the reference classes.

We modify the definitions of precision and recall to consider *pairs* of objects, rather than individual objects. Let n be the number of objects. Then there are $\frac{n(n-1)}{2}$ pairs of objects. Each such pair must fall into one of four categories: the objects are put in the same class by both the reference clusters and the clustering algorithm, they are clustered together but in difference reference clusters, etc.

clustered together?		yes	no
in same reference class?	yes	a	b
	no	c	d

If a, b, c and d are the number of object pairs in each case, then $a+b+c+d = \frac{n(n-1)}{2}$. Precision and recall can now be computed the same way as in standard information retrieval: precision $= a/(a+c)$ and recall $= a/(a+b)$. Other metrics such as F1 or accuracy are defined in the usual way.

Note that there is a simple probabilistic interpretation of these metrics. Precision is equivalent to the conditional probability that two documents are in the same reference class given they are in the same cluster. Recall is equivalent to the conditional probability that two documents are in the same cluster given they are in the same reference class. Let $\mathcal{R}$ denote the event that a document pair is in the same reference class and $\mathcal{C}$ denote that a document pair is in the same cluster. Then we have that precision $= P(\mathcal{R} \wedge \mathcal{C} \mid \mathcal{C}) = P(\mathcal{R} \mid \mathcal{C})$, and recall $= P(\mathcal{R} \wedge \mathcal{C} \mid \mathcal{R}) = P(\mathcal{C} \mid \mathcal{R})$.

Note that precision is biased towards small clusters, but because we are considering document pairs it is not trivially maximised by placing every document in its own cluster. Recall is biased towards large clusters, as it reaches the maximum when all documents are placed in one cluster. Finally, we observe that precision and recall are symmetric, in the sense that precision(A, B) = recall(B, A) for any two clusterings A and B.

4.3 Evaluation

Figures 4.1–4.3 show the precision, recall and F1 scores of the clusters generated by the various algorithms we tried. Zamir's quality measure $Q(\mathrm{C})$ and the normalised mutual information measure $Q(\mathrm{NMI})$ are in our experiments highly correlated with our precision metric, therefore we decided not to list the values for $Q(\mathrm{C})$ and $Q(\mathrm{NMI})$ in our graphs.

Precision is biased towards a large number of small clusters, because it is easier for a clusterer to find a small number of similar services multiple times than to find a large number of similar clusters. Therefore we believe that $Q(\mathrm{C})$ and $Q(\mathrm{NMI})$ are in our experiments in fact dominated by precision, although it is claimed that $Q(\mathrm{NMI})$ is not biased, and although they do not reach their optimal value for singleton clusters.

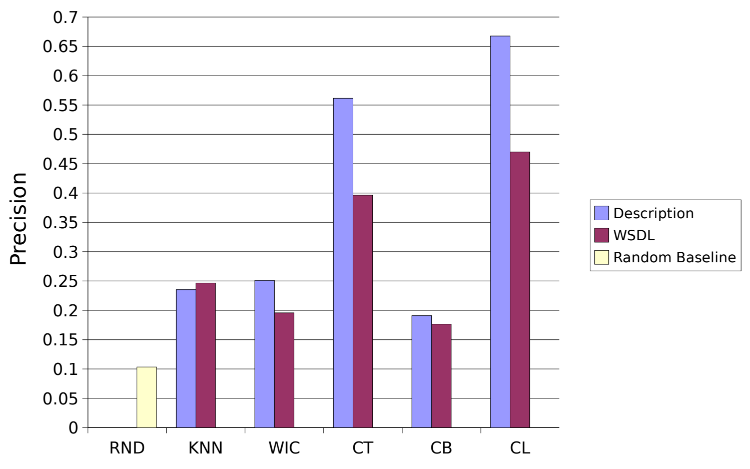
Not surprisingly, none of the algorithms does particularly well, because the Web Services clustering problem is quite challenging. In many cases even humans disagree on the correct classification. For example, SALCentral.org manually organised its Web Services into their own taxonomy, and their classification bears little resemblance to ours. Furthermore, we have 11 categories in our reference classification, which is a rather high number. However, in terms of precision, all our algorithms outperform the random baseline.

All clustering algorithms tend to "over-refine", meaning that they produce far more clusters than classes exist in the reference data. For the centroid based and complete-link algorithms, the number of clusters could be decreased if we set a lower minimum similarity, but especially the centroid based algorithm then tends to produce very large clusters with more than 100 Web Services.

Note that recall is strongly affected by such an over-refinement. In our experiments, it turns out that F1 (see figure 4.3), is largely dominated by recall. However, precision is more important than recall in the application scenarios that motivate our research. Specifically, consider generic automatic data integration scenarios in which Web Services that have been clustered together are then automatically invoked simultaneously. For example, a comparison shopping agent would invoke operations from all Web Services that were clustered into a "E-Commerce" category. Precision is more important than recall for the same reason why precision is more important in today's document search engines: in a Web populated by millions of Web Services, the danger is not that a relevant service will be missed, but that irrelevant services will be inadvertently retrieved.

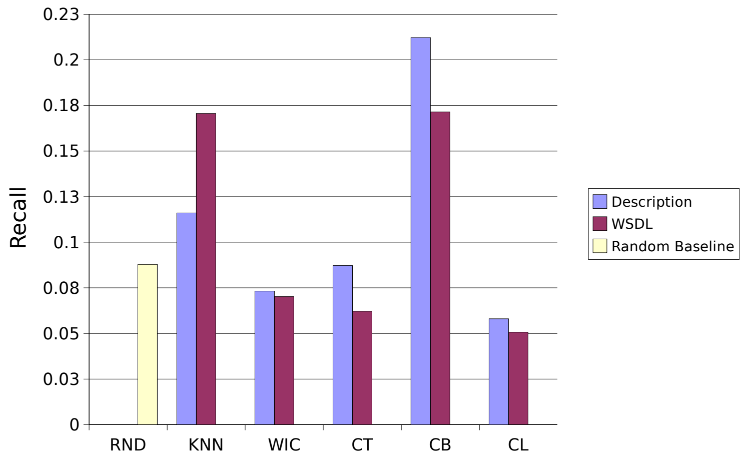
We conclude from these data that Web Service category clustering is feasible based just on WSDL descriptions, through clearly hand-crafted text descriptions (e.g., SALCentral.org's description or text drawn from UDDI entries) produce even better results.

We did not make use of the multi-view split of the dataset for our unsupervised experiments. With the (only very recent) advent of multi-view clustering (Bickel & Scheffer 2004) we believe it will be possible to improve our results in future work.



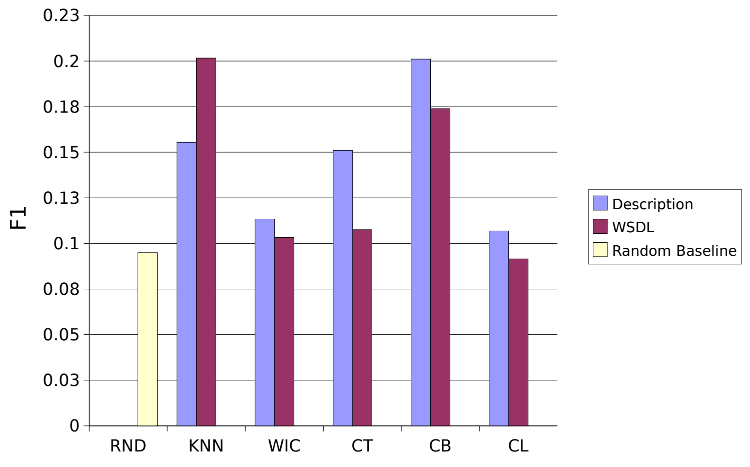
RND: Random baseline, KNN: k-Nearest-Neighbour, WIC: Word-IC, CT: Common Term, CB: Centroid Based, CL: Complete Link

Figure 4.1: *Precision for the various clustering algorithms*



RND: Random baseline, KNN: k-Nearest-Neighbour, WIC: Word-IC, CT: Common Term, CB: Centroid Based, CL: Complete Link

Figure 4.2: *Recall for the various clustering algorithms*

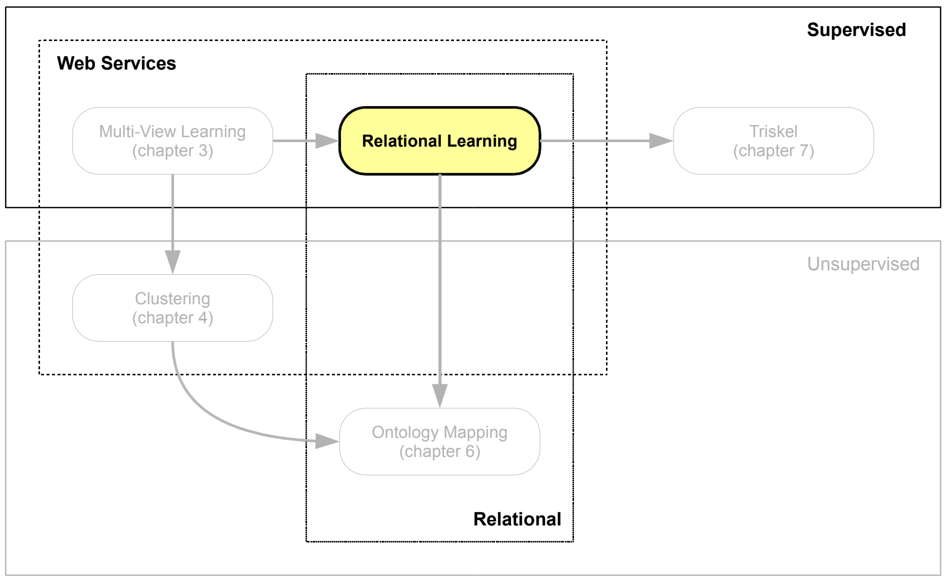


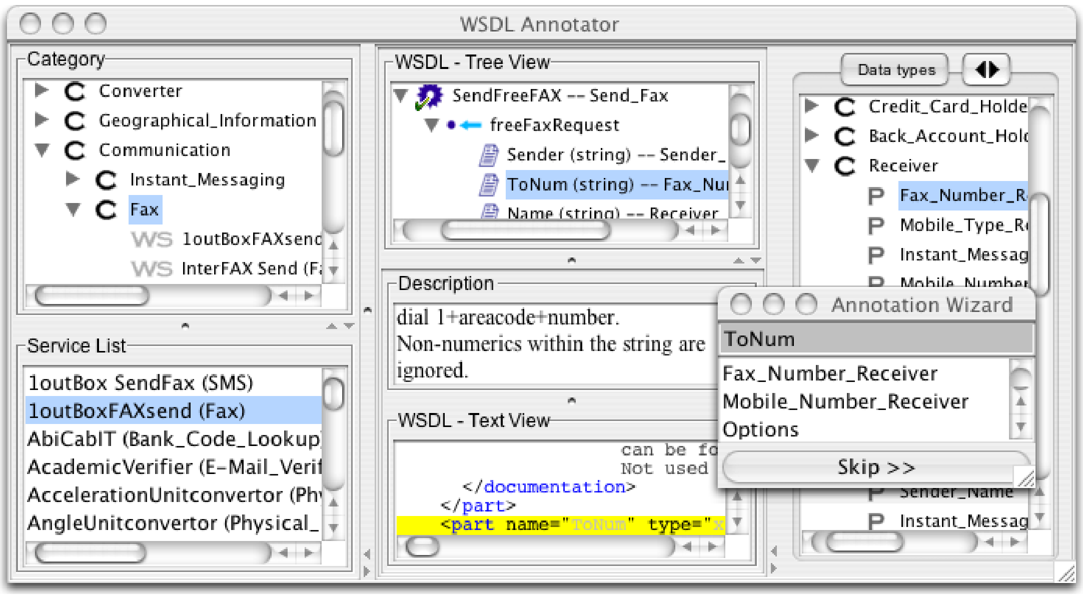
RND: Random baseline, KNN: k-Nearest-Neighbour, WIC: Word-IC, CT: Common Term, CB: Centroid Based, CL: Complete Link

Figure 4.3: *F1 for the various clustering algorithms*

Chapter 5

Annotating Web Services





dial 1+areacode+number. Non-numerics within the string are ignored.

```
                   can be fo
                   Not used
    </documentation>
  </part>
  <part name="ToNum" type="x
```

Figure 5.1: *ASSAM uses machine learning techniques to semi-automatically annotate web services with semantic metadata.*

While the experiments presented in the previous two chapters yielded encouraging results when classifying web services as a whole, classifications for all components of a web service are needed if semantic metadata such as annotations with OWL-S is to be generated automatically. In this chapter, we first present an application for annotating semantic web services that we call ASSAM. Second, an approach for classifying the components of a web service using iterative relational classification is presented. We show how existing iterative relational classification algorithms can be improved using an ensemble. We also present a study for an interactive application that assists a user in annotating semantic web services.

5.1 ASSAM: A Tool for Web Service Annotation

5.1.1 Use Cases

ASSAM[1] is designed primarily for users who want to annotate many similar services. Typically, these will be end users wanting to integrate several similar web services into his or her business processes. But the annotation task might also be performed by a centralised semantic web service registry.

Our tool could also be useful for programmers who are only interested in annotating a single web service they have created.[2] In order to make his or her service compatible with existing services, a developer might want to annotate it with the same ontology that has already been used for some other web services. The developer could import the existing Web Services in ASSAM and use them as training data in order to obtain recommendations on how to annotate his or her own service.

[1]ASSAM is available for download at `http://www.andreas-hess.info/projects/annotator/`
[2]Thanks to Terry Payne who pointed out this use case.

5.1.2 Functionality

Fig. 5.1 shows the ASSAM application. Note that our application's key novelty—the suggested annotations created automatically by our machine learning algorithm—are shown in the small pop-up window.

The left column in the main window contains a list of all web services currently and the category ontology. Web services can be associated with a category by clicking on a service in a list and then on a node in the category tree. When the user has selected a service and wants to focus on annotating it this part of the window can be hidden.

The middle of the window contains a tree view of the WSDL. Port types, operations, messages and complex XML schema types are parsed from the WSDL and shown in a tree structure. The original WSDL file is also shown as well as plain text descriptions from the occasional documentation tags within the WSDL or a plain text description of the service as a whole, such as often offered by a UDDI registry or a web service indexing web site.

When the user clicks on an element in the WSDL tree view, the corresponding ontology is shown in the right column and the user can select an appropriate class by clicking on an element in the ontology view. Currently different ontologies for datatypes and operations are used. At present we allow annotation for operations, message parts and XML schema types and their elements. Port types or messages cannot be annotated, because there is no real semantic meaning associated with the port type or the message itself that is not covered by the annotation of the operations or the message parts.

Once the annotation is done it can be exported in OWL-S. The created OWL-S consists of (see also section 1.1.4):

- a service profile,
- a service model,
- a grounding,
- a concept file, if complex types where present in the WSDL, and
- a service file that imports the above

Note that this also includes XSLT transformations as needed in the OWL-S grounding to map between the traditional XML Schema representation of the input and output data and the OWL representation.

5.1.3 Limitations

Because we do not handle composition and workflow in our machine learning approach, the generated process model consists only of one atomic process per operation. The generated profile is a subclass from the assigned category of the service as a whole – the category ontology services as profile hierarchy. The concept file contains a representation of the annotated XML schema types in OWL-S. Note that it is up to the ontology designer to take care that the datatype ontology makes sense and that it is consistent. No inference checks are done on the side of our tool. Finally, the grounding is generated that also contains the XSLT mappings from XML schema to OWL and vice versa.

For the OWL export, we do not use the annotations for the operations at the moment, as there is no direct correspondence in OWL-S for the domain of an operation. Atomic processes in OWL-S are characterised only through their inputs, outputs, preconditions and effects; and for the profile our tool uses the service category.

5.1.4 Working around the Training Data Problem

A serious limitation is also the fact that any supervised learning algorithm does not work without training data. In the interactive use cases we have in mind for ASSAM, the classifier would learn while the user annotates more and more (parts of) web services, but the user would not have assistance right from the start. To work around this problem, we decided to implement a crude but effective method of "bootstrapping" the classifiers. Before any real training data exists, we simply make predictions based on the class labels. While this is not a generic solution suitable for any machine learning problem, it works well in our case. Since we want to annotate services that will usually have meaningful names with an ontology that will usually also have meaningful names, we can assume that there is a certain overlap. Anecdotal evidence suggests that this bootstrapping approach is useful in an interactive setting. However, the evaluation of our iterative ensemble classification algorithm was done without using the labels.

5.1.5 Related Work in the Web Services Area

As far as we are aware, we were (Heß & Kushmerick 2003) the first to propose the use of machine learning for the task of semi-automatically annotating semantic web services.

Paolucci et al. addressed the problem of creating semantic metadata (in the form of OWL-S) from WSDL (Paolucci, Srinivasan, Sycara & Nishimura 2003). However, because WSDL contains no semantic information and their tool does not use other sources, their software provides just a syntactic transformation. The key challenge – to map the XML data used by traditional web services to classes in an ontology – is not addressed by their tool.

Patil et al. (Patil, Oundhakar, Sheth & Verma 2004) proposed using a combination of lexical and structural similarity measures for the web service matching task. They assume that the user's intention is not to annotate similar services with one common ontology, rather they address the problem of choosing the right domain ontology among a set of ontologies.

As opposed to their approach, Marta Sabou (Sabou 2004) addresses the problem of creating suitable domain ontologies in the first place. She uses shallow natural language processing techniques to assist the user in creating an ontology based on software APIs.

The web service search engine Woogle (Dong, Havey, Madhavan, Nemes & Zhang 2004) allows not only for keyword queries, but also for searching structurally similar operations.

The approach that we present in this chapter ignores a valuable source of information for annotation by restricting itself to the schema only and ignoring the actual values of parameters that are passed on to the service being invoked. Johnston and Kushmerick developed an algorithm that makes use of this data (Johnston & Kushmerick 2004).

5.2 Iterative Relational Classification

For the classification of relational data, iterative classification algorithms that feed back predicted labels of associated objects have been used. In this section we show two extensions to existing approaches. First, we propose to use two separate classifiers for the intrinsic and the relational (extrinsic) attributes and vote their predictions. Second, we introduce a new way of exploiting the relational structure. When the extrinsic attributes alone are not sufficient to make a prediction, we train specialised classifiers on the intrinsic features and use the extrinsic features as a selector. We apply these techniques to the task of semi-automated web service annotation, a task with a rich relational structure.

5.2.1 Iterative Ensemble Classification

Figure 5.2.1 illustrates the relational structure that we use to represent a service. As opposed to the previous task where we only classified services as a whole (compare also figure 3.2), we are now interested in classifying all nodes from the shaded areas in the graph. S, O and D denote these subsets as defined in section 2.3.4.

Our iterative algorithm differs in some points from Neville and Jensen's algorithm. In contrast to (Neville & Jensen 2000), we assume that the object graph has more than one connected component. In Neville and Jensen's experiments, the task was to classify companies as either a bank or a chemical company. As in our model, the companies correspond to vertices in a graph. The relations between the company vertices in their dataset are such that every company is through some intermediate vertices connected to every other company. In our example, we have one connected component for each web service. The services themselves are not connected to each other, but the components within the individual services are connected as illustrated in figure 5.2.1.

Another difference is that in their approach, one single classifier is trained on all (intrinsic and extrinsic) features. We train two separate classifiers, one on the intrinsic features ("$h_{\vec{\mathrm{si}}}$") and one on the extrinsic features ("$h_{\vec{\mathrm{de}}}$"), and vote together their predictions.

Treating the intrinsic and extrinsic views separately promises two advantages: First, we hope to achieve a better overall accuracy when using an ensemble of two classifiers rather than a single classifier trained on all features. The experiments presented in chapter 3 suggest that this might be helpful, when a discriminative classification algorithm is used. The second advantage is obvious for an iterative relational setup: The

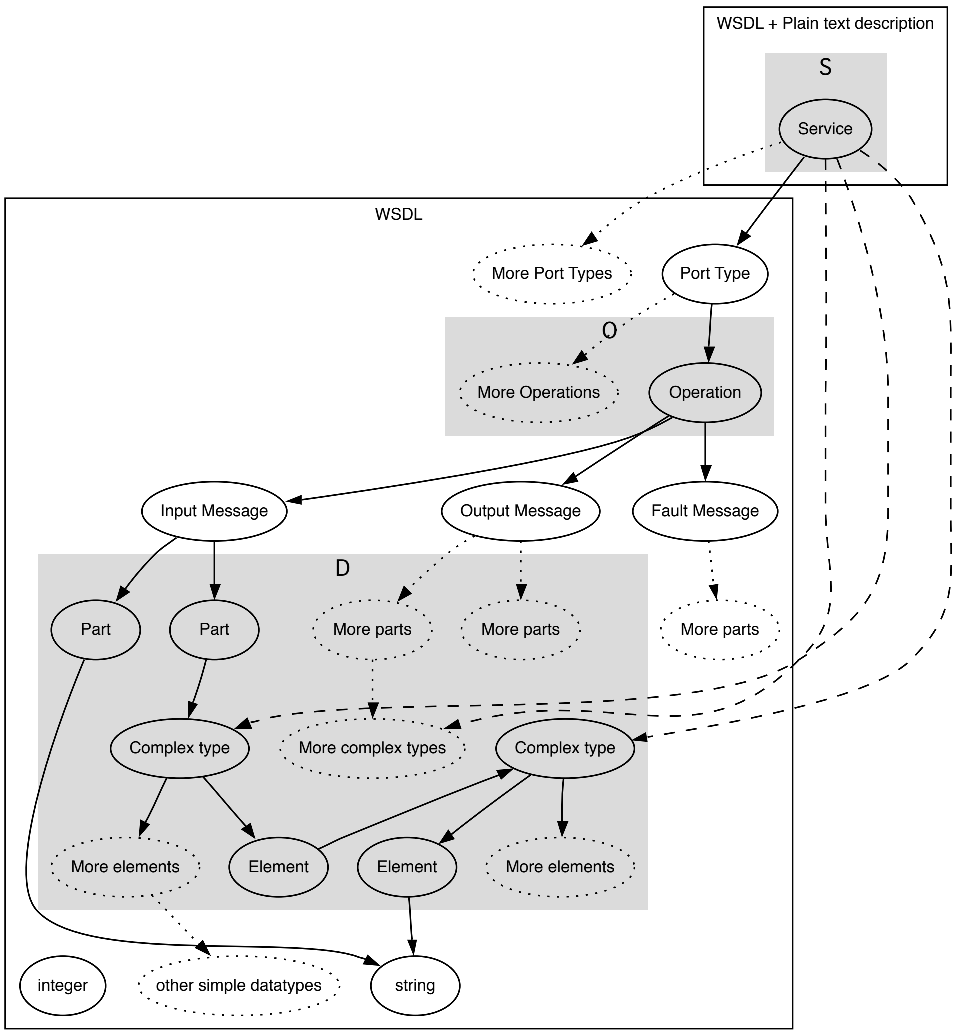


Figure 5.2: *A web service as a graph. The shaded boxed (S, O and D) denote the subsets of vertices that we want to classify. Compare to figure 3.2, where the task was to classify services as a whole.*

classifier cannot be mislead by missing features in the first iteration, when the extrinsic features are yet unknown. The classifier trained on the extrinsic features is simply not used for the first pass. In subsequent iterations we include extrinsic features that are based on the class labels predicted in the previous round. The classification process is repeated until a certain termination criterion (e.g. either convergence or a fixed number of iterations) is met.

When using a Naive Bayes classifier, however, we cannot expect to get a different result from a split approach than from a single classifier. In our setup, we combine the predictions of the two classifiers by multiplication. A Naive Bayes classifier estimates the probability of an instance belonging to a class with a term of the form $P(C)\prod_{i=1}^{n} P(F_i|C)$ (see also section 2.4.3). Because the prior probability is the same for both splits, a combination of two Naive Bayes classifiers would calculate the probability as $P(C)^2(\prod_{i=1}^{m} P(F_i|C))(\prod_{j=m+1}^{n} P(F_j|C))$ (assuming that we split the features at some index m). The only difference to the original equation is that we over-emphasise the prior probability. We cannot expect the results to improve.

5.2.2 Specialised classifiers

In the general case (i.e. if we use another classifier than Naive Bayes) the split approach, where the $h_{\vec{\text{si}}}$ and $h_{\vec{\text{de}}}$ classifiers are combined using multiplication, works best, when both views would by themselves be sufficient to make a prediction. This intuition is the same as behind Co-Training (see section 2.4.5 or Blum's and Mitchell's paper on Co-Training (Blum & Mitchell 1998)). However, in more challenging prediction tasks, the extrinsic view alone gives additional evidence, but is not sufficient to make a prediction. In our web services dataset, this is the case on the datatype level. The fact that a web service belongs to the "book selling" category is by itself not sufficient to classify its input parameters, but the information is still useful. We therefore introduce a second mode for incorporating the extrinsic features: We train a set of classifiers on the intrinsic features of the components, but each of them is only trained on the subset of the instances that belong to one specific class. In our setting, we train specialised classifiers on the datatypes level on all instances that belong to the same category[3].

To formally specify these specialised classifiers, we first define what we mean by a specialisation. We take a very generic view: a specialisation on a set $X \subset V$ (see section 2.3.4) is a subset $\Phi_X(c) \subset X$ of a subset of all vertices that consists of all elements that satisfy some constraint c. In our setup, we use related entities to define the specialisation constraints. Consider the set of parameters D. One specialisation Φ_D might be "all parameters of services that are in the 'book selling' category." Let $b \in S'$ be the concept in the taxonomy that represents the "book selling" category and $l \in L$ be the arc label that represents the relation from a parameter to the service it belongs

[3]To avoid over-specialisation, these classifiers are in our experiments actually not trained on instances from a single category, but rather on instances from a complete top-level branch of the hierarchically organised category taxonomy. Note that this is the only place where we make use of the fact that the class labels are organised as a hierarchical taxonomy. We do not do any further inference or reasoning.

Algorithm 5 Iterative Ensemble Classification

```
for $v \in V$ do
  $\vec{\mathrm{di}}_{\mathrm{int}}(v) \leftarrow$ predictions made by $h_{\vec{\mathrm{si}}}(v)$
end for
/* Initially, use predictions obtained from intrinsic view only */
$\vec{\mathrm{de}}(v) \leftarrow \bigoplus_{x \in \mathrm{rel}(v)} \vec{\mathrm{di}}_{\mathrm{int}}(x)$
for a fixed number of iterations do
  for $v \in V$ do
    $\vec{\mathrm{di}}_{\mathrm{ext}}(v) \leftarrow$ predictions made by $h_{\vec{\mathrm{de}}}(v)$ or $h_c(v)$ based on $\vec{\mathrm{de}}(v)$.
    /* Vote predictions obtained from intrinsic and extrinsic view */
    $\vec{\mathrm{di}}(v) \leftarrow \vec{\mathrm{di}}_{\mathrm{int}}(v) \otimes \vec{\mathrm{di}}_{\mathrm{ext}}(v)$
  end for
  $\vec{\mathrm{de}}(v) \leftarrow \bigoplus_{x \in \mathrm{rel}(v)} \vec{\mathrm{di}}(x)$
end for
return $\forall v \in V : \vec{\mathrm{di}}(v)$
```

to. Then we write:

$$\Phi_D(b) = \{x | \exists y \in \mathrm{rel}(x, l) \wedge \mathrm{map}_{\mathrm{tax}}(y) = b\}$$

We require that all specialisations be mutually exclusive and exhaustive. Let $\phi_x(c, y)$ $= 1$ if $y \in \Phi_X(c)$ and 0 otherwise. Mutually exclusiveness and exhaustivity means simply that $\forall y : \sum_c \phi_X(c, y) = 1$.

The specialisation $\Phi_X(c)$ is used to define a set of training data for learning a specialised classifier $h_{X,c}$. Specifically, $h_{X,c}$ is trained on the intrinsic features $\vec{\mathrm{si}}(x)$ for every training instance $x \in X$ in $\Phi_X(c)$.

To avoid biasing the algorithm too strongly, we still combine the results of the $h_{X,c}$ classifier with the $h_{\vec{\mathrm{si}}}$ classifier in each iteration. For each level we use either the $h_{\vec{\mathrm{de}}}$ or h_c classifier, but not both. We chose the h_c method for the datatypes and the $h_{\vec{\mathrm{de}}}$ method for the category and the domain.

Algorithm 5 explains our iterative classification ensemble, and fig. 5.3 illustrates its classification phase. In fig. 5.3 and algorithm 5, let n denote the number of object components; $\vec{\mathrm{si}}_X$ the static intrinsic and $\vec{\mathrm{de}}_X$ the dynamic extrinsic features; $h_{\vec{\mathrm{si}}}$ and $h_{\vec{\mathrm{de}}}$ the intrinsic and extrinsic classifiers, and h_c denotes a specialised classifier as discussed above. $\vec{\mathrm{di}}_{\mathrm{int}}$ denotes the predictions made by $h_{\vec{\mathrm{si}}}$ and $\vec{\mathrm{di}}_{\mathrm{ext}}$ denotes the predictions made by the extrinsic classifier $h_{\vec{\mathrm{de}}}$ or h_c.

5.2.3 Related Work

Several variations of link-based text classification have been proposed. The classical task here is to classify web pages connected by hyperlinks. Lu and Getoor address this task in (Lu & Getoor 2003). They cast the problem as predicting the classes of objects or nodes in a graph considering links between objects or edges in the graph. However, in their model only one flavour of objects exists: the classes for all objects are drawn from only one ontology. Iterative classification algorithms were also used

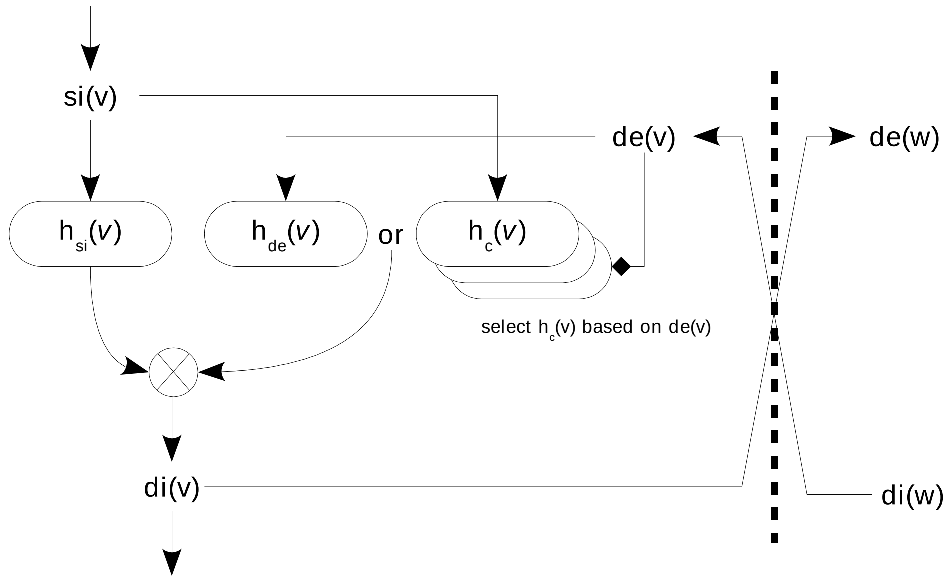


Figure 5.3: *Iterative Ensemble Classification.*

by Chakrabarti (Chakrabarti, Dom & Indyk 1998) before. We already mentioned the approach by Neville and Jensen (Neville & Jensen 2000).

An approach that is very similar to the specialised classifiers described here is used by Finn and Kushmerick (Finn & Kushmerick 2004) for information extraction. They use a two-level learning approach. At the first level (L1), each token in a document is predicted either as the start of a text fragment to extract, or not. L1 is trained on all available data. A second classifier (L2) is then used to identify the end of a fragment, given that its start was predicted by L1. L2 is trained on just the subset of the training data, namely a window of tokens near field starting positions. In our terminology, the L2 classifier is a specialised classifier h_c, and the L1 predictions are used to decide when to invoke L2, just as we use extrinsic features to select the appropriate h_c.

5.3 Parameters

In an iterative feed-back setting such as ours, a sensible choice of all parameters is both crucial and rewarding: An improvement in one of the classifiers can lead to an improvement in other classifiers as well. In some respects the parameters we discuss here are relevant not only to our specific setting, but are common to all iterative classification algorithms. The important parameters are:

1. Structure: Which layers of the model are used for feedback and how?
2. Voting: How are the results of the different views combined?
3. Termination criterion: What number of iterations is optimal?

4. Inference, Constraints: Use information from the class ontologies?
5. Features: Use binary or continuous features?
6. Classification algorithms: Which one to use?

5.3.1 Structure

One might expect that this structure is inherent to the problem, but this is only partly true. In our web services dataset, we have identified three layers of semantic metadata, as described in section 2.3.4 and shown in figure 5.2.1: the category of the service as a whole, the domain of the service's operations and the datatypes of its input and output parameters. This leaves a variety of possible choices how the feed-back is actually performed. One might say that the category of a service is determined by the domain of its operations, and the domain of the operations is determined by the datatypes of its input and output parameters. But it is also possible to see it the other way around: The datatypes of an operation's parameters are dependent on the operation's domain, which is dependent on the service's category. Instead of choosing either a top-down or bottom-up approach, our framework also allows for modelling both dependencies at the same time.

Preliminary tests showed that it is not guaranteed that using as many extrinsic features as possible automatically yields the best results. It is best to choose between either the $h_{\vec{\mathrm{de}}}$ or h_c classifier based on the nature of the extrinsic view: If the extrinsic view alone provides enough information to make a reliable prediction, it is best to combine the extrinsic and intrinsic view in an ensemble way. If the extrinsic view alone is not sufficient for a reliable classification, it is better to use the h_c classifier. There are, however, two points that are important when considering the h_c classifier: Using a set of specialised classifiers means that there are as many classifiers as there are distinct class labels in the extrinsic view and each classifier is only trained on a subset of the available training data. To have a set of many classifiers can not only be computationally expensive, it also means that the number of training instances for each classifier is only a fraction of the total number of training instances.

We used *static* extrinsic features on the domain and datatype level by incorporating text from children nodes: Text associated with messages was added to the text used by the operations classifier, and text associated with elements of complex types were added to the text used by the datatype classifier classifying the complex type itself. Preliminary experiments have shown that the accuracy of the classifier is increased if we use static extrinsic features here. Note that this appears to contradict our earlier results from chapter 3, where we claimed that simply adding text from child nodes does not help. In the experiments in chapter 3, we were classifying on the category level only, and the bag of words for the domain and datatype classifiers consisted of text for all operations/datatypes in that service. In the experiments discussed in this chapter, the classifier for the operations and datatypes classify one single operation or parameter

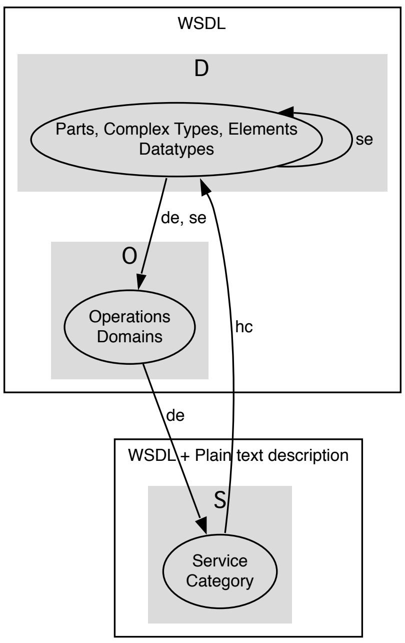


Figure 5.4: *Feedback structure actually used in experiments*

only. Thus, the amount of text is much smaller. We conclude from these results that there is in fact a trade-off: If the initial amount of text for one bag of words is small, it is better to add more text. If we can assume that the distribution of words in different bags is independent, it is better to keep them separate and use multiple classifiers.

Given these arguments for and against the different modes of incorporating the extrinsic evidence and some empirical preliminary tests, we eventually decided to use the setup shown in figure 5.3.1. In the figure, “se” denotes the use of static extrinsic features by incorporating text, “de” denotes the use of dynamic extrinsic features and “hc” denotes the use of specialised classifiers.

5.3.2 Combining Static and Dynamic Features

One of the fundamental differences between our approach and the original algorithm by Neville and Jensen is the way the different views—i.e., the intrinsic and extrinsic features—are combined. Neville and Jensen's approach is to add the extrinsic features as additional attributes in a stacking-like fashion and train a single classifier on all features. Neville and Jensen tested their algorithm on a low-dimensional dataset. On our dataset, however, this way of combining the evidence of the static and dynamic features did not work. In section 5.4 we also report results for a setup that combines static and dynamic features in this “non-ensemble” way. We conclude that this is due to the fact that both the static and the dynamic features are high-dimensional, and that the initial noise in the dynamic features strongly affects the overall performance of the classifier.

5.3.3 Number of Iterations

The iterative classification is not guaranteed to converge, and it is also not guaranteed that more iterations always produce better results, as the predictions might get stuck in a local maximum. We tried several fixed numbers of iterations in preliminary experiments, and eventually decided to terminate the classification process after five iterations.

5.3.4 Inference and Constraints

In our web services dataset as well as in many other relational learning tasks, the class labels are not arbitrary but rather they are organised as an ontology. This ontology could be a simple hierarchical structure, but it could also convey more complex constraints. In our datatypes ontology, it would be possible to model restrictions on the relations between complex types and their elements. For example, a complex type that represents a book in the datatypes ontology can only have elements that are assigned a class label that represents a property of book in the ontology.

In the present setting we do not make use of constraints that can be inferred from the ontology, with one exception. As mentioned in section 5.2, we use the hierarchy in the services ontology to prevent over-specialisation of the h_c classifiers. However, we believe that the use of inference that can be derived from the ontologies where the class labels are drawn from could increase classification accuracy. However, we leave this aspect for future work.

5.3.5 Features

The well-known range of choices for representing words in a bag-of-words model as features in a classifier includes binary features, term frequency and TFIDF. In our case, we also have to choose a suitable representation for the extrinsic features. We can choose between a single feature denoting the most common linked class or use a feature vector with one element per possible class. When using the latter, the feature vector can be binary or frequency based. It is also possible to use the complete ranked distribution or the top n of classes as output by the classifiers instead of just the absolute predictions.

In our experiments, using a single feature for the extrinsic view does not make much sense, as we are using a separate classifier for the dynamic view. In preliminary experiments, using the complete ranked distributions did not work. We believe that this is due to the fact that in conjunction with the base classification algorithm we used and the one-vs-all setting, the absolute confidence values in the ranked distribution are too close together, and that then combining the distributions for all linked objects introduces too much noise.

We decided to use a binary feature vector with one attribute per possible linked class for the dynamic features. For the static features, we decided to use binary features as well. In section 5.2, we already described another method of incorporating dynamic extrinsic features, the specialised classifiers, and the possibility to add terms from linked document parts as a way to use static extrinsic features.

5.3.6 Classification algorithms

Due to the large number of classifiers and evaluations in our iterative framework, it is desirable to use a fast classification algorithm, especially if the intended application requires user interaction and low response-times are necessary.

In our implementation we are using the Weka library of classifiers (Witten & Frank 1999) off the shelf. For our experiments we chose the HyperPipes algorithm in a one-against-all configuration, as it offered a good tradeoff between speed and accuracy. We did not use the SMO algorithm because it turned out to be computationally too expensive. We decided to cancel preliminary experiments because they took too long.

5.4 Evaluation

We evaluated our algorithm using a leave-one-service-out methodology. We compared it against a baseline classifier with the same setup for the static features, but without using the dynamic extrinsic features.

To determine the upper bound of improvement that can be achieved using the extrinsic features, we tested our algorithm with the correct class labels given as the extrinsic features. This tests the performance of predicting a class label for a document part when not only the intrinsic features but also the dynamic features (the labels for all other document parts) are known. Following (Neville & Jensen 2000), we refer to this upper bound as the "ceiling".

We also compared it against a non-ensemble setup, where the extrinsic features are not added using a separate classifier but rather are just appended to the static features. Classification is then done with a single classifier. This setup closely resembles the original algorithm proposed by Neville and Jensen. Again, the same set of static features was used.

In the evaluation we ignored all classes with one or two instances, such as occurred quite frequently on the datatype level. The distributions are still quite skewed and there is a large number of classes. There are 22 classes on the category level, 136 classes on the domain level and 312 classes on the datatype level. Our corpus consists of 164 services with a total of 1138 annotated operations and 5452 annotated parameters.

Fig. 5.5 shows the accuracy for categories, domains and datatypes. The setups in the figure are as follows:

Baseline In this setup, all classifiers are trained only on intrinsic features.

Ensemble In this setup, both intrinsic and extrinsic features are used. A separate classifier is used for the extrinsic features and the results are then combined. Note that the extrinsic features are based on the initial intrinsic classification. Thus, the extrinsic features may contain noise, if the initial predictions are wrong

Ceiling This setup is similar to the ensemble setup, but the extrinsic features are always noise-free because they are based on the correct class label. This setup serves as

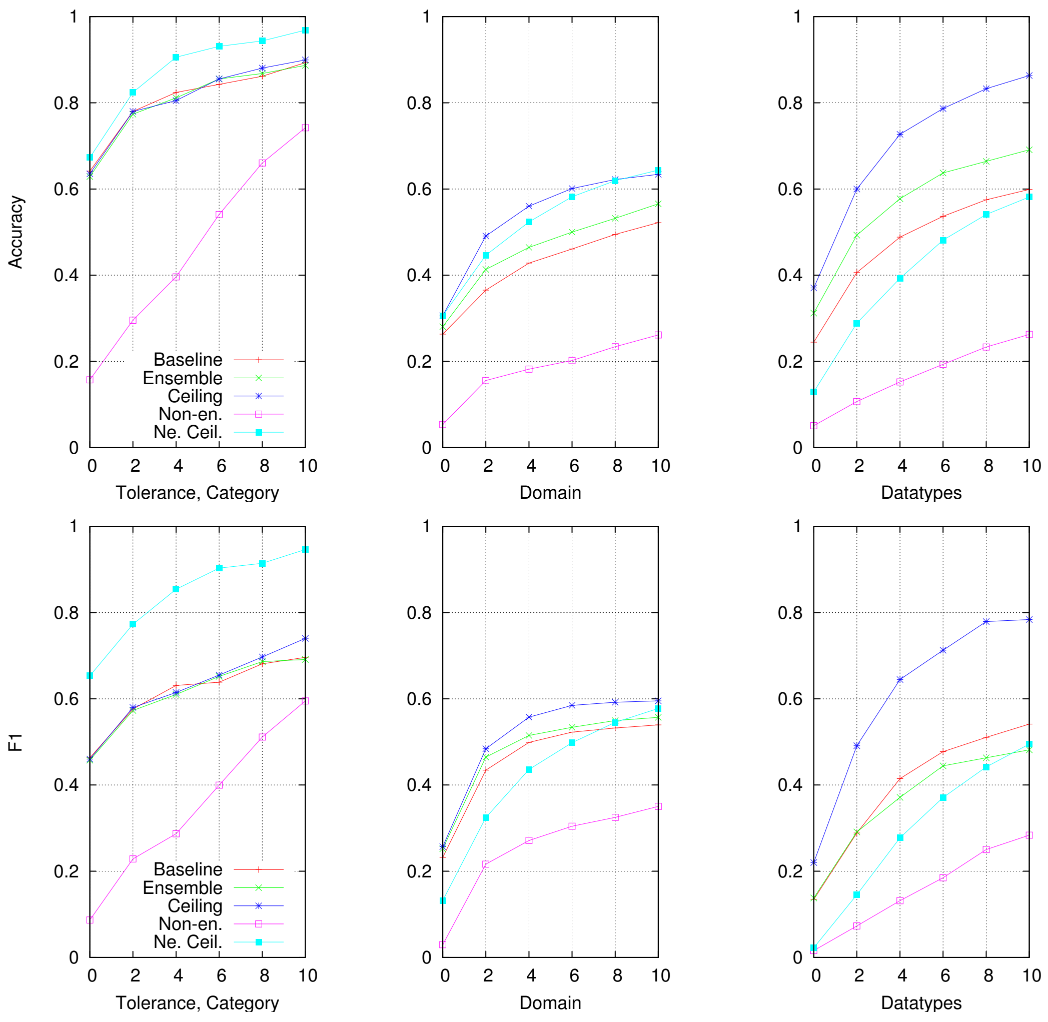


Figure 5.5: *Accuracy and macroaveraged F1 on the three ontologies.*

an upper baseline, because it shows the maximum improvement in accuracy that we can achieve by using extrinsic features.

Non.-en. In this setup, we also use intrinsic and extrinsic features, but only a single classifier is trained on a feature vector that is a combination of all intrinsic and extrinsic features.[4]

Ne. Ceil. As above, only a single classifier is trained on a feature vector that is a combination intrinsic and extrinsic features. The extrinsic features are noise-free. This is an upper baseline for the non-ensemble configuration.

As mentioned earlier, in mixed-initiative scenario such as our semi-automated ASSAM tool, it is not necessary to be perfectly accurate. Rather, we strive only to ensure

[4]Note that this configuration performs quite badly here. We conclude that, since we are using a discriminative classifier on a high-dimensional feature set, the algorithm fails to learn correct bounds if the features are a combination of both intrinsic and extrinsic features. We would, however, expect this configuration to perform very similar to the ensemble setup, if a Naive Bayes classifier would be used instead of the HyperPipes algorithm. Because we combine the predictions of the intrinsic and extrinsic classifiers my multiplication, the only difference would be that the prior distribution is more important in the ensemble setup, because it appears twice in the equation for the posterior probability.

that the correct ontology class is in the top few suggestions. We therefore show how the accuracy increases when we allow a certain tolerance. For example, if the accuracy for tolerance 9 is 0.9, then 90% of the time, the correct prediction is within the top 10 of the ranked predictions. Note that on the datatypes level, macroaveraged F1 (as shown in Fig. 5.5) for the iterative ensemble is worse than for the baseline while accuracy is still above the baseline. This is due to the fact that at this level of tolerance the baseline has a higher recall for many small classes. Macroaveraged precision is still higher for the iterative ensemble.

Note that on the category level incorporating the additional evidence from the extrinsic features does not help. In fact, for some tolerance values the ceiling accuracy is even worse than the baseline.

Also, we could not achieve good results with the non-ensemble setup (denoted as "NE"). This setup scored worse than the baseline. For the datatypes, even the ceiling accuracy (denoted as "NE Ceiling") was below the baseline.

We evaluated the statistical significance of the accuracy improvement of our algorithm compared to the baseline on the datatype and domain level. The improvement is on both levels statistically significant with $p < 0.05$ according to a Student t-test.

5.5 Conclusion

5.5.1 Summary

We have demonstrated two extensions to prior research on iterative classification algorithms for relational tasks. First, we have shown that in a high-dimensional environment such as text classification it is better to use separate classifiers for the intrinsic and extrinsic features and vote their predictions rather than to use one classifier trained on both types of features. Second, we have introduced a new mode for relational classification where the extrinsic features serve as a selector for a specialised intrinsic classifier trained on a subset of the training instances. We have applied these techniques to the domain of semi-automatically annotating semantic web services, and shown that it outperforms conventional approaches.

5.5.2 Future Work

As mentioned, we believe that incorporating domain knowledge in the classification process can increase the overall performance. In future work we will investigate the use of inference over the ontology and how it can help the machine learning algorithm. In particular, currently we ignore the hierarchical structure of the class labels, but we will explore whether this structure can be exploited to improve prediction accuracy. On the application side, we will continue to improve our WSDL annotator tool. We believe that tools such as ours are needed to bootstrap the semantic web and semantic web services, and that machine learning is a very helpful technique that should be applied here.

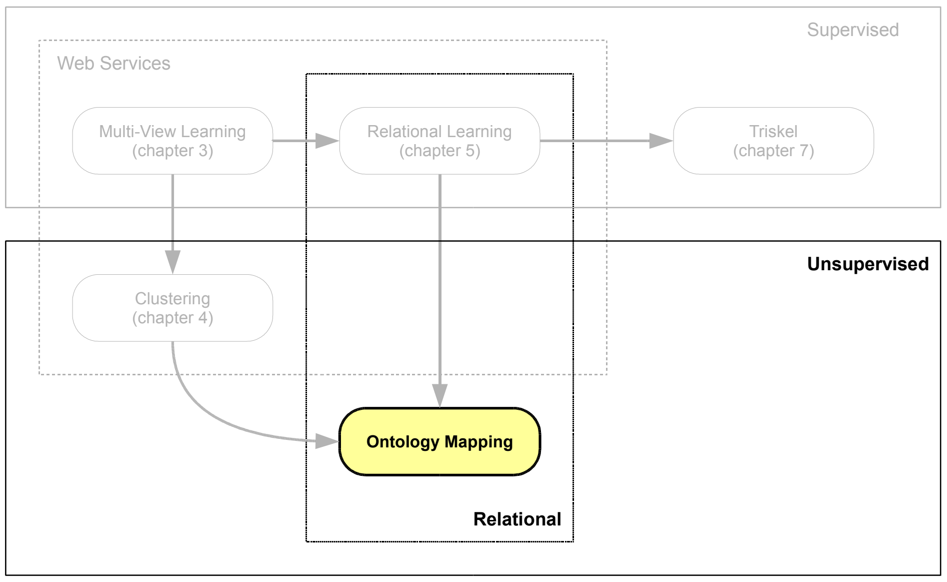
The work of Sabou (Sabou 2004) that has been mentioned above is complementary to the work presented in this chapter. A joint approach seems promising and will be discussed in future work.

At present, the author of the thesis has moved on to work on the WS-Diamond project[5]. The goal of this project is to produce a framework for self-healing complex web service workflows. In this context, there will be a need of additional metadata in order to make the workflow diagnosable. A sub-project of WS-Diamond will therefore tackle the problem of semi-automatically creating such metadata. This subproject can be seen as a continuation of the work presented in this chapter, where we focused on annotation and ignored issues such as composition and workflows.

[5] `http://wsdiamond.di.unito.it/`

Chapter 6

Aligning Ontologies



While the iterative ensemble techniques presented in the previous chapter are supervised and thus require training data, it is also possible to adapt the iterative algorithm to work in an unsupervised way. The task of ontology alignment is an interesting problem field to use our algorithms on. It is both a more general task than Web Service annotation as well as a task that has been addressed by unsupervised algorithms before. In this chapter, we present an unsupervised iterative ensemble approach to ontology mapping. We compare the performance of our algorithm with the performance of other alignment algorithms. Each of these algorithms has its own strengths and weaknesses, and our algorithm can compete well against the current state-of-the-art.

6.1 Problem Formulation

The ontology alignment problem consists of finding a number of mappings $v' = \text{map}(v)$ for as many $v \in V, v' \in V'$ as possible, see definition 2 in section 2.3.

We restrict ourselves to finding mappings for classes and properties only. We define the set of classes as $C \subset V$ resp. $C' \subset V'$ and the set of properties as $P \subset V$ resp. $P' \subset V'$ (see section 2.3.4), and we only map classes to classes and properties to properties. Furthermore, we restrict the mapping function from definition 2 to finding one-to-one-mappings, i.e. being injective, but not necessarily surjective.

We split the problem into two parts: First, we implement a similarity function *sim* according to definition 3 (see section 2.3) and use this similarity function to compute all pairwise similarities between all $v \in V$ and $v' \in V'$. Section 6.3 describes our solution to this problem.

Second, we treat these pairwise similarities as a bipartite graph $B = (V + V', E)$ with the entities from V and V' as nodes and a weighted edge where the similarity between two entities $\text{sim}(v, v') > 0$. The problem of obtaining the *map*-relation is then equivalent to the problem of finding a matching in this bipartite graph. Recall that according to definition 3 it is not required that $\text{sim}(v, v') = \text{sim}(v', v)$. In that case, the edges in the bipartite graph B are directed and the weights are not symmetric. However, this is only a minor aspect of the problem. Section 6.4 describes the application of two well-known graph-algorithms to this problem.

6.2 Related Work

While many approaches have been proposed for schema matching[1] in the past (see section 1.1 for a number of references), dedicated algorithms for ontology matching are newer. Among these newer algorithms are NOM (Ehrig & Sure 2004) and QOM (Ehrig & Staab 2004) (the latter of which is optimised for speed), the interactive PROMPT (Noy & Musen 2003), Euzenat et al's. integrative proximity measure (Euzenat & Valtchev

[1]Schema or ontology mapping as discussed here is – although often cast as a graph matching problem – not to be confused with the (NP-hard) largest subgraph isomorphism problem, see e.g. (Veale 1998) for a discussion. It is worth noting that lexical similarity plays an important role in ontology mapping and that finding isomorphic subgraphs is neither sufficient nor required to address this problem.

2003) and the more recent OLA (Euzenat, Loup, Touzani & Valtchev 2004), which combine a variety of different similarity measures. The level of competition that came along with these different approaches has led to ontology alignment contests. Such contests have taken place at the Information Interpretation and Integration Conference (I^3CON) in 2003 and the Third International Workshop on Evaluation of Ontology Based Tools in 2004 (Sure, Corcho, Euzenat & Hughes 2004). In section 6.6 we will compare our own algorithm to those presented in (Sure et al. 2004).

Ehrig and Staab in (Ehrig & Staab 2004) identified a structure common to most ontology alignment algorithms. Basically, this common structure boils down to an iterative computation of one or more (intrinsic) similarity measures between entities and the application of structural (extrinsic) similarity measures. Our algorithm adheres to that structure, too. However, there are two features which make it distinct from all other algorithms that we are aware of.

The first point where our algorithm differs from others is the way in which extrinsic similarity is computed. In a variety of approaches, extrinsic similarity is basically just the propagated intrinsic similarity of the neighbouring entities. In our approach, we compute extrinsic similarity by using a feature vector. Section 6.3.2 describes the details.

The second novel feature is the way in which the similarities are transformed into mappings. We are not aware of other algorithms that treat the problem as maximal weighted matching.

6.3 The Algorithm

The structure of our algorithm follows closely the structure of algorithm 5 for supervised iterative ensemble classification presented in section 5.2. The main difference is of course that for our unsupervised ontology mapping algorithm there is usually no training data and we cannot just plug in any learning algorithm. However, this can be overcome by substituting a distance metric for a learning algorithm.

Again, we distinguish between *intrinsic*, *static extrinsic* and *dynamic extrinsic* features. Analogous to the supervised setting, the intrinsic features of a concept in the ontology are features that are inherent to the concept. We use the following intrinsic features:

1. The local name of the concept URI
2. The label of a concept, if one exists
3. Comments, if they exist

As in section 5.2 and as in (Neville & Jensen 2000), we define static extrinsic features as features that are derived through a concept's relation to other concepts, but do not change as we make more accurate predictions about the mapping. In our approach, we consider the following static extrinsic features:

1. Text derived from individuals of a class
2. Text derived from values of properties in individuals

In our algorithm, we treat static extrinsic features as intrinsic features. For the remainder of this section, if we talk about intrinsic features, we mean both intrinsic and static extrinsic features. If we talk about extrinsic features, we mean dynamic extrinsic features only.

The (dynamic) extrinsic features are based on the relation of the concept to other concepts. Since the structure of an ontology is more general than the rather fixed structure of a web service, we have to consider several types of relations. The possible relations depend on the expressiveness of the ontology language used, however, the algorithm itself is independent of the concrete syntax. For our implementation we assume OWL (Dean, Schreiber, Bechhofer, van Harmelen, Hendler, Horrocks, McGuinness, Patel-Schneider & Stein 2004) as the ontology language, and our terminology follows the OWL terminology. We are considering the following relations between concepts:

1. Superclasses
2. Subclasses
3. Superproperties
4. Subproperties
5. Defined properties for a class
6. Domain of a property
7. Range of a property
8. Siblings of classes and properties

6.3.1 Computing Intrinsic Similarity

In our implementation, we use distance metrics from the well-known SecondString library[2] for the intrinsic features. We conducted experiments with the Jaro-Winkler metric (Winkler & Thibaudeau 1991, Jaro 1976, Jaro 1989) and a version of Levenshtein edit distance (Levenshtein 1965) that is scaled to the range 0..1 for comparing labels and local names. We used a soft-token metric (Cohen et al. 2003) with Jaro-Winkler resp. scaled Levenshtein edit distance as the base string distance metric.

We also experimented with a similarity based on WordNet.[3] We used a similarity metric based on Euzenat's implementation in the OWL alignment API (Euzenat 2004). We decided, however, not to use it in the current setup. Preliminary experiments suggested that on many datasets no or only a marginal improvement can be achieved.

[2]`http://secondstring.sourceforge.net/`, see also (Cohen, Ravikumar & Fienberg 2003)
[3]`http://wordnet.princeton.edu/`

This small benefit is, however, contrasted by a much greater computational effort. It may be possible to overcome these limitations by using a more sophisticated algorithm for computing a semantic similarity based on WordNet. This is, however, deferred to future work.

To determine the overall intrinsic similarity between two concepts, we compare both the local names and labels against each other using Jaro-Winkler or Levenshtein and comments and text from instances using the respective soft-token metric and use the maximum value. To avoid overemphasising small similarities, we disregard similarities that are smaller than a threshold of 0.4 and map similarities greater than 0.4 to the full range $[0, 1]$. Let v be a concept in ontology V and v' be a concept in ontology V'. If we look at the computation of the similarity between these two concepts, then each similarity value (before applying the threshold) is denoted as $\text{sim}_i(v, v')$ with i ranging from 1 to the number of different metrics n. The overall intrinsic similarity between two concepts is denoted as $\text{sim}_{\text{int}}(v, v')$ and is determined as follows:

$$\text{sim}_{\text{int}}(v, v') = \max_{i=1}^{n} \left(\begin{Bmatrix} 0, & \text{if } \text{sim}_i(v, v') < 0.4 \\ (\text{sim}_i(v, v') - 0.4)/(1 - 0.4), & \text{otherwise} \end{Bmatrix} \right) \tag{6.1}$$

6.3.2 Computing Extrinsic Similarity

The main difference between our approach and existing schema matching algorithms is the way extrinsic similarity is computed. In previous approaches extrinsic or structural similarity is usually propagated through a graph structure that is determined by the schema or ontology (see the section on related work). This is based on the assumption that two nodes are similar if their neighbours are similar.

In our approach, we derive an extrinsic feature vector $\vec{\text{de}}(v)$ from the relations rel$v$, see definition 14. Each element in the vector corresponds to a concept in the ontology. If an element is 1 it means that the corresponding concept is related to v. To use this vector to determine the extrinsic similarity to another concept v', we need to first compute initial similarities (based on the intrinsic features) for all related objects of v. These similarities $\text{sim}(x, y'), x \in \text{rel}(v), y' \in V'$, translate into *dynamic intrinsic features* of the related objects as per definition 12 and are then combined into the dynamic extrinsic feature vector $\vec{\text{de}}(v)$ as per definition 14. Note that the elements in $\vec{\text{de}}(v)$ are based on the relations of $v \in V$, but its elements correspond to vertices in V', which is necessary in order to compute the similarity of vector $\vec{\text{de}}(v)$ to known relations in V'.

The discussion about the feature vector in our supervised iterative ensemble learning approach for web service classification (see section 5.3.5) applies here as well. By using only the best mapping map(v) for each object and using a binary feature vector, we choose the same representation scheme here.

Note that the similarities that can be computed based on this vector are not symmetric. Since the feature vector is based on the best mapping for each concept, the fact that v maps to v' does not necessarily mean that the best mapping for v' is v, if the

Algorithm 6 Iterative Ensemble Mapping

```
for $v \in V$ do
  $\vec{\text{di}}_{\text{int}}(v) \leftarrow [\text{sim}_{\text{int}}(v, v'_0), \text{sim}_{\text{int}}(v, v'_1), \ldots, \text{sim}_{\text{int}}(v, v'_{|V'|-1})]$
end for
/* Initially, use predictions obtained from intrinsic view only */
$\vec{\text{de}}(v) \leftarrow \bigoplus_{x \in \text{rel}(v)} \vec{\text{di}}_{\text{int}}(x)$
for a fixed number of iterations do
  for $v \in V$ do
    $\vec{\text{di}}_{\text{ext}}(v) \leftarrow [\text{sim}_{\text{ext}}(v, v'_0), \text{sim}_{\text{ext}}(v, v'_1), \ldots, \text{sim}_{\text{ext}}(v, v'_{|V'|-1})]$
    /* Vote predictions obtained from intrinsic and extrinsic view */
    $\vec{\text{di}}(v) \leftarrow \vec{\text{di}}_{\text{int}}(v) \otimes \vec{\text{di}}_{\text{ext}}(v)$
  end for
  $\vec{\text{de}}(v) \leftarrow \bigoplus_{x \in \text{rel}(v)} \vec{\text{di}}(x)$
end for
return $\forall v \in V : \vec{\text{di}}(v)$
```

overall similarity $\text{sim}(v, v')$ is greater than the similarity of v to all other $x' \in V'$ but less than the similarity $\text{sim}(v', x)$ of v' to some $x \in V$.

6.3.3 Iterative Algorithm

The iterative algorithm for computing the overall – intrinsic and extrinsic – similarities is essentially the same as algorithm 5. The only difference is that in algorithm 5 we were assuming a Machine Learning algorithm h, while for the problem of ontology mapping we just assume an arbitrary similarity function. Algorithm 6 formally specifies our mapping algorithm.

Note that we are not restricted to finding mappings from V to V'. Since we do not have training data here, which would be for one direction of mappings only, we can compute the mappings from V' to V in just the same way. Recall that because of the way we compute the extrinsic similarity these mappings are not necessarily equal. The next section explains how we can use this asymmetry.

6.4 Postprocessing Steps

Once we have computed the overall similarities, we have to compute the actual one-to-one mapping. This is the problem of finding a matching in a bipartite graph. A bipartite graph is a graph where the nodes can be split in two groups such that every edge connects two nodes from both groups. Every similarity that has been calculated in the previous step corresponds to a weighted edge in such a bipartite graph.[4] A matching in a graph is a set of edges such that no node is incident to more than one edge. In our setting this corresponds to a one-to-one mapping: For every instance in one ontology we want to find one instance in the other ontology. A matching is called maximum-weighted, if there is no other matching where the sum of all edge weights in the matching

[4]Note that this bipartite graph must not be confused with the graph interpretation of the two ontologies!

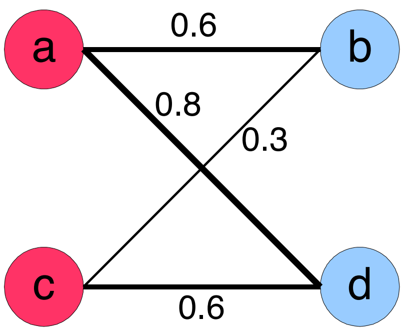


Figure 6.1: *A Bipartite Graph. The matching* $\{(a, b), (c, d)\}$ *is maximal, but* {*(a,d),(c,b)*} *is a stable marriage.*

is bigger. Looking for a maximum-weighted matching is a straightforward way to come up with a good solution to our original problem to find a one-to-one mapping.

Consider the bipartite graph in figure 6.1. The matching $\{(a, b), (c, d)\}$ is maximal (the sum of the weights is 1.2), but it is *unstable*. To explain what this means it is best to resort to an analogy: Imagine the nodes in our graph were men and women, let us use the names Alice, Bob, Carol and Dave for the a, b, c and d nodes. Between each of these men and women there is a well-defined level of attraction that is represented by the weight of the edge that connects them. A couple is said to be married if there is an edge in a matching that connects them. If Alice marries Bob and Carol marries Dave, the overall happiness in this scenario is maximised. Their marriages, however, are not stable for an obvious reason: Alice and Dave like each other more than their current spouses, so they will break up with their partners and elope. But if Alice marries Dave, and Carol marries Bob, the marriages are stable. Carol likes Dave more than her husband Bob, but Dave is very happy with Alice and will not break up his marriage. The same applies to Bob and Alice.

Gale and Shapley have shown in (Gale & Shapley 1962) that for graphs with an equal number of "men" and "women" a stable marriage can always be found and presented an $O(n^2)$ algorithm to compute such a stable marriage that is male-optimal (and female-pessimal). In (Irving, Leather & Gusfield 1987) an $O(n^4)$ algorithm is presented that computes an overall-optimal stable marriage. Note that the Gale/Shapley algorithm does not assume that the weights of the edges are symmetric. For the reasons mentioned in section 6.3.2, the weights are not symmetric in our case. Because the number of vertices in V and V' is not necessarily equal, a perfect match (in the graph-theoretic sense) is not always possible. It is therefore necessary to modify the termination criterion of the original Gale/Shapley algorithm slightly: A "man" is only put back into the set of unmarried men, if there still exists a "woman" that he likes better than zero, otherwise he remains unmarried. Algorithm 7 illustrates the Gale/Shapley algorithm with the modified termination criterion.

Melnik et al. in (Melnik et al. 2002) propose to compute either a stable marriage

Algorithm 7 Gale/Shapley algorithm for stable marriages

```
$U \leftarrow V$
$U' \leftarrow V'$
$L(u, u') \leftarrow$ weight of the edge $(u, u')$ in $B$
$L'(u', u) \leftarrow$ weight of the edge $(u', u)$ in $B$
$M \leftarrow \emptyset$
**while** $U \neq \emptyset$ **do**
  $u \leftarrow$ random element from $U$
  $u' \leftarrow x'$ such that $L(u, x')$ is maximal
  **if** $u' \in U'$ **then**
    /* $u'$ still "unmarried" */
    remove $u'$ from $U'$
    remove $u$ from $U$
    insert $(u, u', L'(u', u))$ into $M$ /* $u$ and $u'$ now "engaged" */
  **else**
    **if** $L'(u', u) > e|(x, u', e) \in M$ **then**
      /* weight of edge $(u', u)$ greater than weight of edge from previous engagement
      */
      remove $(x, u', e)$ from $M$ /* Break old engagement of $x$ and $u'$ */
      **if** $\exists u'|L(x, u') > 0$ **then**
        /* Only re-insert $x$ into $U$, if there still exists a possible match $u'$. */
        insert $x$ into $U$
      **end if**
      insert $(u, u', L'(u', u))$ into $M$ /* $u$ and $u'$ now "engaged" */
    **else**
      **if** $\exists u'|L(u, u') > 0$ **then**
        /* Only re-insert $u$ into $U$, if there is still exists a possible $u'$ for $u$. */
        insert $u$ into $U$
      **end if**
    **end if**
  **end if**
  $L(u, u') \leftarrow 0$ /* $u$ cannot "propose" to the same $u'$ twice */
**end while**
**return** $M$
```

or the maximum weighted matching to find a good mapping. We compared the two approaches empirically on our data. We used an off-the-shelf implementation of James Munkres' algorithm (Munkres 1957) (also referred to as "Hungarian" algorithm) to compute maximum-weighted matchings. The problem of computing maximum-weighted matching in bipartite graphs is also known as the assignment problem. Munkres' algorithm computes an exact solution to the assignment problem in $O(n^3)$ time. However, more efficient (e.g. (Galil 1986)) or parallel (e.g. (Bertsekas & Castanon 1990)) algorithms. Furthermore, it computes only *one* maximum-weighted matching, (Fukuda & Matsui 1992) shows an algorithm that can find *all* maximum-weighted matchings. As opposed to the Gale/Shapley algorithm, Munkres' algorithm is not suited for bipartite graphs with directed edges and asymmetric weights. Therefore, we created new bipartite graphs with undirected edges that have the sum of the weights of the directed edges as weights.

6.5 Parameters

Our matching algorithm as presented in this chapter has various parameters. Some of these parameters have been discussed in section 5.3, where we described our iterative ensemble algorithm for supervised learning.

1. Structure: What relations in the ontology are suitable to serve as extrinsic features?
2. Termination criterion: What number of iterations is optimal?
3. Inference, Constraints: Use additional information from the ontologies?
4. Post-processing: Stable marriage or maximum-weighted matching?
5. Apply post-processing step in between iterations?
6. Intrinsic similarity: Which string-distance metric?
7. Extrinsic features: Binary or continuous?
8. Thresholds: Suppress mappings with low confidence?

6.5.1 Structure

Depending on the expressiveness of the underlying ontology language, several relations between classes or properties are defined. We enumerate the relations that we considered for our algorithm in section 6.3. Preliminary experiments suggested that the incorporation of siblings as extrinsic features did not improve the results. We decided to use two different setups for our final experiments: First, a setup where we used all the relations listed in section 6.3 except for siblings. In our evaluation, we call this configuration "dublin2". Second, we considered a setup where only the subsumption relations on classes and properties were considered. In this setup, we ignore the domain and range of a property as well as declared properties for classes as extrinsic features. We denote this configuration as “dublin3”.

6.5.2 Number of Iterations

As in our experiments with iterative ensemble classification, we decided to use a fixed number of iterations as termination criterion for reasons of simplicity, and because it is not proven that the algorithm converges. Preliminary empirical experiments suggested that the algorithm is not very sensitive to the exact number of iterations. We set the number of iterations to five, the same number we used for the experiments described in chapter 5.

6.5.3 Inference

When mapping rich ontologies, it is sometimes possible to exploit knowledge drawn from the ontologies to impose constraints on the mappings or to infer mappings. Although we believe that for some mapping tasks exploiting such knowledge could increase the mapping accuracy, such an approach is out of scope of this thesis. We restrict ourselves to using the information obtained through the iterative relational algorithm to compute the final mappings. The set of ontologies we used for evaluating our algorithm does not have a very rich structure, so in comparison with other algorithms that may use such inference, our algorithm has no disadvantage.

6.5.4 Post-processing

As discussed above, we have to consider at least two ways of creating a mapping from the acquired similarities, if we demand a one-to-one mapping. We can compute either a stable marriage or a maximum weighted matching. In our empirical experiments, we tried both approaches. In the graphs and tables presenting our results we denote configurations that use the Gale/Shapley algorithm (as opposed to a maximum weighted matching) with the letter “g”.

We also tried both possible answers to the question when the post-processing step should be applied. We denote the configurations where we applied the post-processing step also in between iterations with the letter “e”. In the other experiments, the post-processing step (i.e. applying the Gale/Shapley or Hungarian algorithm) was only performed after the iteration phase of the algorithm has been completed.

6.5.5 Intrinsic Similarity

We already discussed the way we compute the intrinsic similarity between two concepts above in section 6.3.1. However, we could plug an arbitrary string distance metric in our framework.

A great variety of string distance metrics – established algorithms as well as ad-hoc measures – is available off-the-shelf in libraries such as the already mentioned SecondString. As mentioned above, we considered the Jaro-Winkler and Levenshtein metrics. Preliminary experiments have shown that with our data, a scaled version of the Levenshtein metric works generally better than Jaro-Winkler. Therefore, we decided to use only the scaled Levenshtein metric in our final experiments. We set the threshold for the soft-token metric to 0.9, i.e. two tokens that have a string similarity greater or equal than 0.9 are considered the same. For other ontologies, however, a different string distance might perform better. The suitability of different string distance metrics for several tasks has been extensively discussed in literature, e.g. (Cohen et al. 2003).

6.5.6 Extrinsic Features

We described a general way for deriving an extrinsic feature vector from the mappings of related concepts in section 6.3.2, see also section 2.3.6 in the introduction. The design choice here is which similarity metric to use. If we use the binary similarity function sim_{bin} as per definition 4, the extrinsic feature vector becomes binary. It is, however, also possible to use a continuous feature vector, if we use another sim function resp. the intermediate result for $\vec{\text{di}}(v)$ from algorithm 6. In our experiments we decided to use a continuous feature vector with the hope that a numerical representation of the similarity would represent the uncertainty about the mapping. Preliminary results suggested that this hope was fulfilled as the results with the continuous feature vector were slightly better than with the binary vector.

6.5.7 Thresholds

In order to avoid spurious mappings it makes sense to use a cut-off value. In the ontology mapping scenario, it is not guaranteed that for some concept in one ontology a concept in another ontology actually exists. In these cases, not making a prediction is the correct answer. But also in other cases it is in several scenarios useful not to make a prediction at all rather than making a bad prediction. For example, consider a semi-automated setting where a human annotator has to review suggestions made by the algorithm.

Furthermore, within our iterative algorithm itself a cut-off value for the extrinsic features can be used. The intuition is that, especially when binary extrinsic feature vectors are used, the algorithm could easily be misled by spurious predictions. On the other hand, if continuous extrinsic feature vectors are used, a cut-off value seems less important, since predictions with a low confidence automatically get a lower weight.

For the precision/recall-graphs, we varied the threshold between 0 and 1 in steps of 0.05. When comparing the different configurations of our algorithm, we used a zero threshold. When comparing our algorithm to other algorithms from the EON 2004 contest, we also report the results for a threshold of 0.5. For comparison with the algorithms from the OAEI 2005 contest we report the results for a threshold of 0 as submitted to the OAEI organisers.

In the "dublin3" setup, we used a threshold of 0.2 for the extrinsic features. In the "dublin2" setup, we did not use a threshold for the extrinsic features. The fact that there is only little difference in the results from the "dublin2" and "dublin3" configuration confirms the intuition that the threshold between iterations is not as important as other parameters, when continuous extrinsic feature vectors are used.

6.6 Evaluation

We evaluated our algorithm on the benchmark ontologies from the 2004 EON Ontology Alignment Contest[5] in order to make some qualitative statements about strengths, weaknesses and most importantly the effect of different parameter settings on the performance. We also participated in the 2005 Ontology Alignment Evaluation Initiative (OAEI 2005, (Euzenat, Stuckenschmidt & Yatskevich 2005)), the follow-up event of the 2004 contest. Most of the benchmark ontologies consist of versions of a base ontology, where different aspects have been changed. Some of these ontologies can be regarded as "sanity checks" rather than real challenges. For example, if we match two ontologies and one of them consists of classes and properties where as the other ontology is the exact same, except for the fact that all properties are suppressed, any sensible algorithm should still match the classes correctly based on the identical labels as well as the subsumption relation. Our algorithm as well as all of the other algorithms in the contest achieve near-perfect results on these tasks. Six ontologies, however, are more interesting: In two cases (denoted as ontologies 205 and 206), all names and labels have been replaced with synonyms or foreign words, and in four cases, independently developed "real-world" ontologies that describe the same domain have been used (301-304). We concentrate on these six ontologies for our evaluation.

We tested various configurations of our algorithm and compared the results from these different setups against each other as well as against the published results from the other participants of the contest. The experiments were conducted in order to answer the four basic (groups of) questions:

1. Do we get any benefit from the extrinsic features as opposed to using the intrinsic similarity only? Should we use all available extrinsic features or is the subsumption relation enough?

2. Is it better to compute the maximum weighted matching or is a stable marriage more important? Should we apply this step only after all similarities are computed, or also between iterations?

3. What threshold is optimal?

4. How does our algorithm perform compared to other algorithms in literature? What are the strengths and weaknesses?

[5]`http://oaei.inrialpes.fr/2004/Contest/`

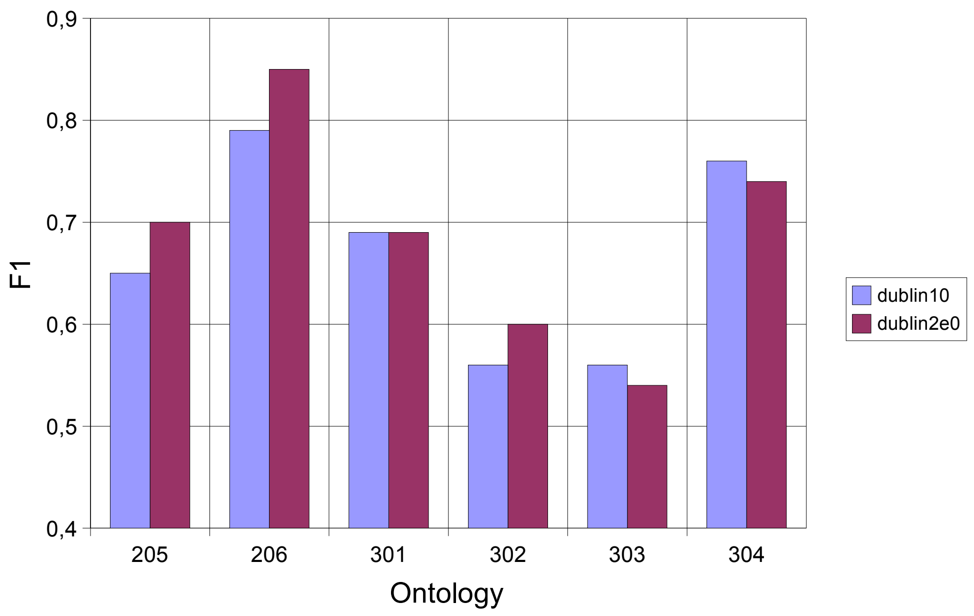


Figure 6.2: *Comparison of a configuration that uses intrinsic features only ("dublin1") and a configuration that also uses extrinsic features ("dublin2e").*

It is important to note that in most of the experiments the difference in performance between the different configurations was quite low. It is therefore hard to draw strong conclusions from the outcome of these experiments, although there are clearly trends visible. However, what the experiments clearly show is that the overall accuracy of ontology mapping is based largely on the initial intrinsic (lexical) mapping. Unfortunately, for other algorithms that also use both lexical and structural similarity, it is rarely published what the contributions of the extrinsic and intrinsic similarities are.

6.6.1 Extrinsic vs. Intrinsic Features

The first question is of course the most crucial one: Is the way in which we use the additional relational information, that differs from other methods known in literature, useful? Does it work? To answer this question, we compared the "dublin1" setup with the "dublin2e" setup. The "dublin1" setup uses only intrinsic features, "dublin2e" uses extrinsic features (in five iterations) as well. Both setups compute a maximum-weighted matching, the "dublin2e" configuration also does this in between operations to compute the extrinsic features.

The results in figure 6.2 (note that the scale starts with 0.4 to emphasise the difference between the two configurations) show that on four ontologies the configuration that uses extrinsic features performs better or equal than the configuration with only the intrinsic features. However, in two of the "real-world" ontologies, using the extrinsic features makes the overall performance worse. The reason for this is that the ontologies 303 and 304 are structurally different from the base ontology and our algorithm is mislead by this structural difference. In that case, any attempt to make predictions

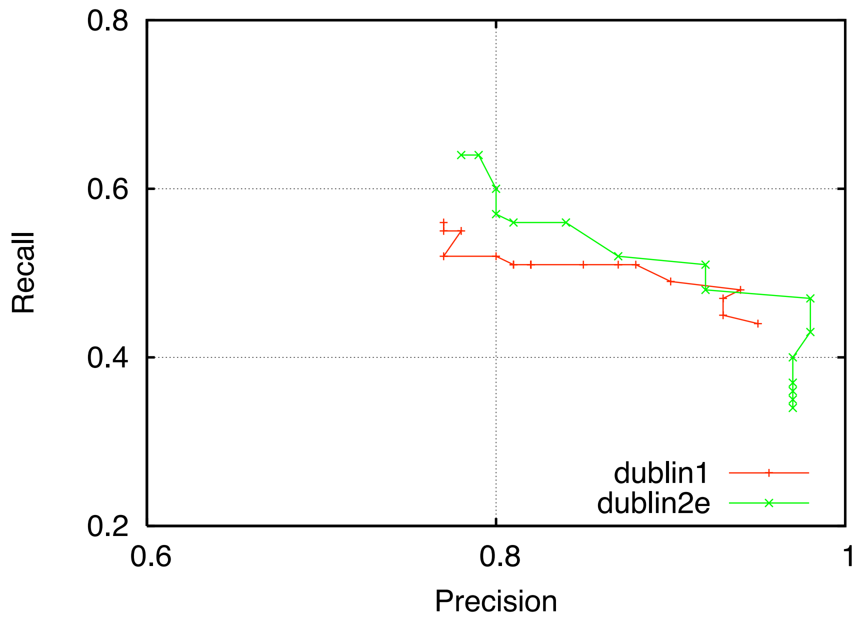


Figure 6.3: *Precision/Recall-Curve for ontology 205 with two different configurations of our algorithm*

based on the structure must fail. The other four ontologies, especially 205 and 206, are structurally quite similar to the base ontology. Here using the extrinsic features helps. We conclude from these results that using relational features can improve the performance, but only if the ontologies that are to be matched are not structurally different. Figure 6.3 shows how precision and recall change for the "dublin1" and "dublin2e" configurations if we vary the threshold. The x-axis measures precision and the y-axis measures recall, the optimal matching algorithm would produce a point in the top right corner. Note that the "dublin2e" curve is mostly above the "dublin1" curve. We conclude that, if we are handling an ontology where using the extrinsic features helps, the benefit from the extrinsic features is independent from the threshold.

6.6.2 Stable Marriage vs. Maximum-Weighted Matching

As far as we are aware, most other current algorithms do not explicitly compute stable marriages or maximum-weighted matchings to determine a one-to-one mapping. The Similarity Flooding algorithm (Melnik et al. 2002) is a notable exception. We compared two configurations that both use extrinsic features in five iterations. The only difference between the two setups is that "dublin2g0" uses the Gale/Shapley algorithm to compute a stable marriage while "dublin20" computes a maximum-weighted matching. Both configurations use no threshold.

Figure 6.4 clearly shows that it is better to compute a maximum-weighted match-

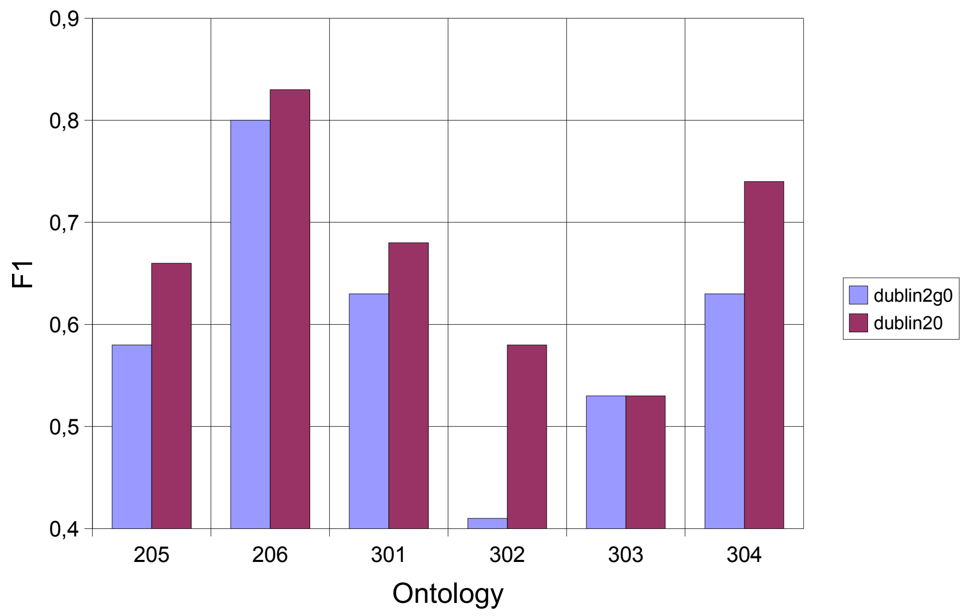


Figure 6.4: *Comparison of a setup with a stable marriage ("dublin2g0") with a configuration with a maximum-weighted matching ("dublin20").*

ing. This setup outperforms the stable-marriage configuration in all but one cases, where there is a tie between both setups. Figure 6.5 shows the precision/recall-curve for ontology 301 with four different configurations of our algorithm. Note that the curves for the maximum-weighted matching setups ("dublin1" without and "dublin2" with extrinsic features) are above the curves for the stable-marriage configurations ("dublin1g" and "dublin2g"). We conclude that the maximum-weighted matching configuration outperforms the stable-marriage configuration regardless of the threshold.

6.6.3 Threshold

As already discussed, the improvements we gain from the extrinsic features and from using maximum-weighted matching instead of stable marriage are independent from the threshold. Figures 6.3 and 6.5 also show the tradeoff between precision and recall. However, to find out what value for the threshold is best, we took a closer look at ontologies 205 and 303. Figures 6.6 and 6.7 show the relation between the threshold and the precision, recall and F1 measures on ontologies 205 resp. 303. Note that varying the threshold has a quite different effect on the two ontologies. In ontology 205, recall drops faster than precision increases. The maximum F1 is reached at a threshold of 0.05. In ontology 303, precision and recall at threshold 0 are lower than in ontology 205. When raising the threshold, recall drops only slightly while precision increases rather quickly. Maximum F1 is reached at a threshold between 0.7 and 0.85. We have to conclude that the best cut-off value for our mapping algorithm depends strongly on the dataset. On the "real world" ontologies 301–304 the threshold is higher, while for the artificial benchmark ontologies the best F1 is reached at a very low threshold.

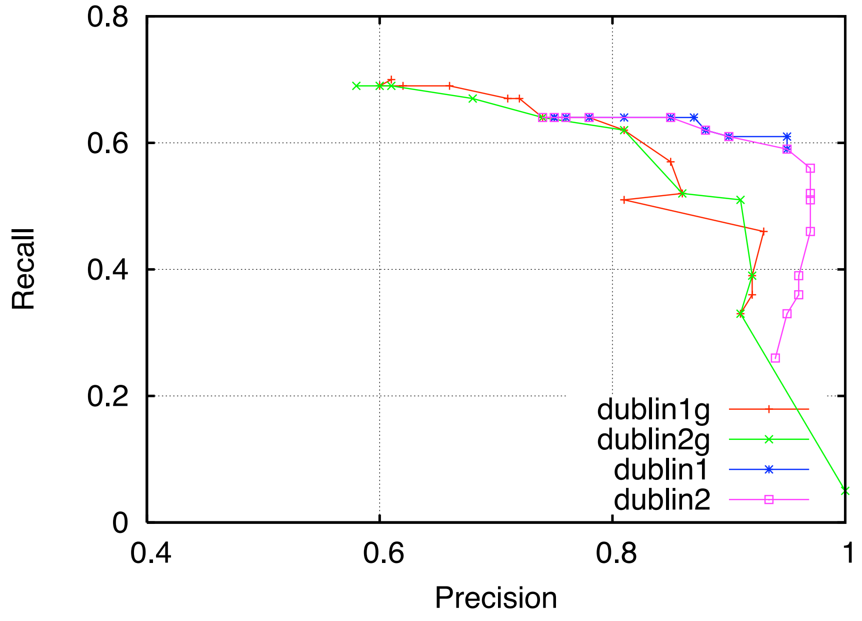


Figure 6.5: *Precision/Recall-Curve for ontology 301 with four different configurations of our algorithm*

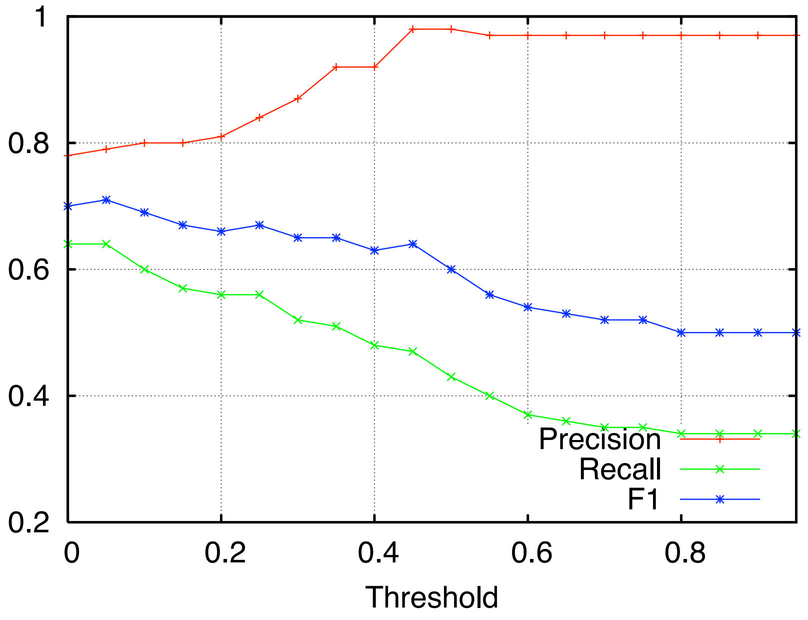


Figure 6.6: *Relation between threshold and precision, recall and F1 for ontology 205*

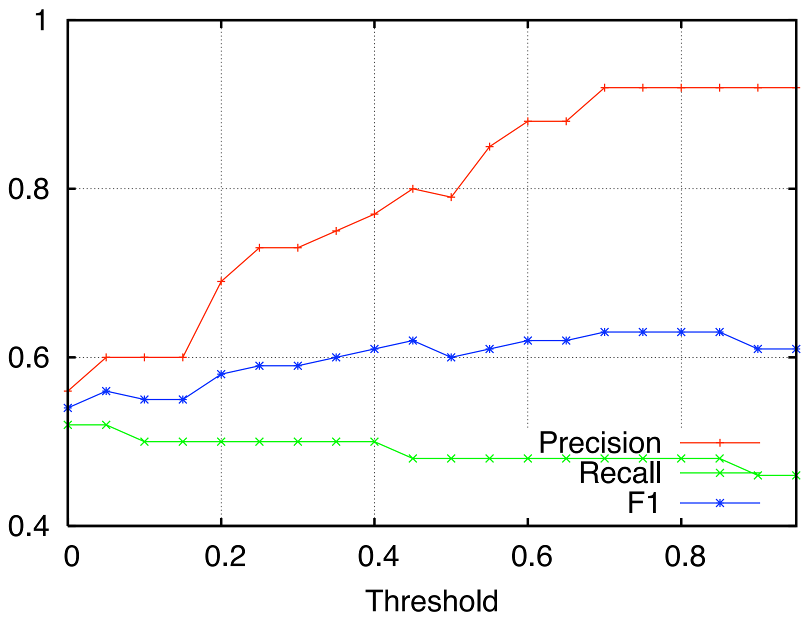


Figure 6.7: *Relation between threshold and precision, recall and F1 for ontology 303*

6.6.4 Comparison with other Algorithms

To evaluate our own method, we compared our results against the published results from the EON 2004 ontology mapping contest and participated in the 2005 Ontology Alignment Evaluation Initiative ((Euzenat et al. 2005)).

EON 2004 Ontology Mapping Contest

In 2004, the algorithms developed at Stanford (Noy & Musen 2003) and Fujitsu (Hoshiai, Yamane, Nakamura & Tsuda 2004) performed best in the contest. Figure 6.8 shows how our algorithm compares to these two algorithms. We used the "dublin2e" configuration with two different thresholds (0, denoted as "dublin2e0" and 0.5, denoted as "dublin2e50") for comparison.

Our configurations can compete well with the Stanford and Fujitsu algorithms. Stanford performs best on the last three ontologies, but is outperformed by our algorithm on the first three datasets. On the other hand, the Fujitsu algorithm outperforms our method on ontologies 301 and 304, but is outperformed by our algorithm on 206 and 302. On ontology 205 there is a tie between our algorithm and Fujitsu's. On the first two ontologies, where the structure of both the base ontology and the test ontology are the same, our algorithm has an advantage. If the structure becomes less important as especially on the last ontology, our algorithm has a disadvantage. The diagram also underlines our statement from the previous section: For ontologies that are structurally

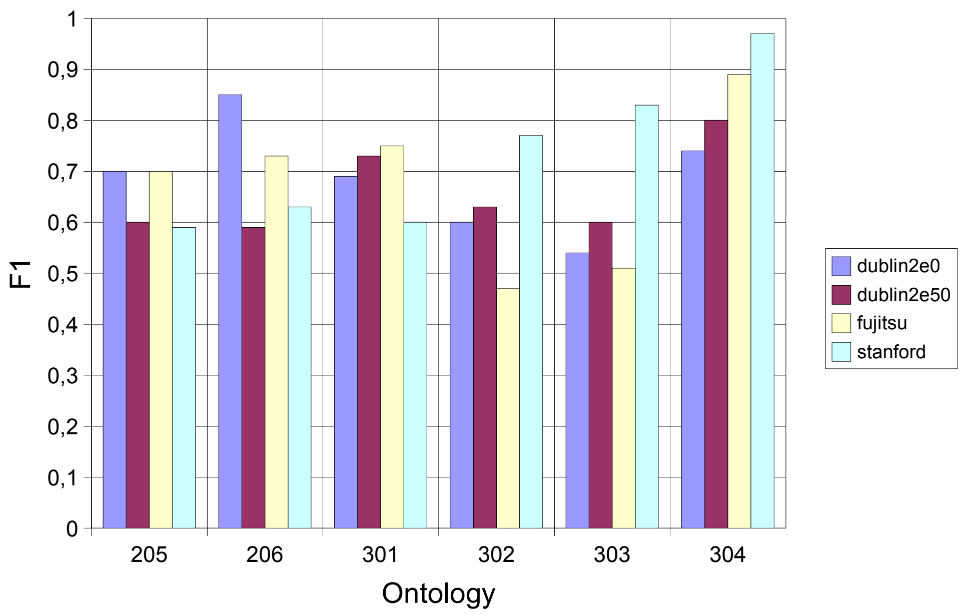


Figure 6.8: *Comparison of two configurations of our algorithm and two other algorithms*

similar, a low threshold is better, for the "real world" ontologies, our algorithm performs better if we use a higher cut-off value.

Ontology Alignment Evaluation Initiative 2005

For the 2005 alignment contest, the organisers added several new benchmark ontologies based on the 2004 benchmark tests. Furthermore, two new real-world tasks were introduced. The first new task consisted of computing an alignment between two taxonomies constructed from the Google, Yahoo and Looksmart web directories. Because the full taxonomy would be very large, the task was split into about 2000 snippets. The ontologies for this task consist only of classes and subsumption relations. There are no properties or individuals. The second new task was to compute an alignment between two ontologies from the medical domain. These ontologies are both very large.

The algorithm that performed best in the 2005 contest was the "Falcon" algorithm by the Southeast University of Nanjin. Our own algorithm can, however, compete well with the "FOAM" algorithm developed in Karlsruhe and the "OLA" algorithm. "Edna" is a simple algorithm that is based on edit distance of the labels and was included by the organisers of the contest as a baseline. Table 6.1, taken from (Euzenat et al. 2005), shows the results of the different algorithms on the benchmark ontologies. Figure 6.9 shows the F1 score. To aggregate the results of the individual tests, the organisers of the contest calculated the precision and recall over all mappings of all test.

Our algorithm in the "dublin20" setting as submitted to the organisers of the OAEI 2005 performs second best after Falcon. From the 2004 algorithms, the one from Stanford has a higher average precision, but a lower average recall than ours.

Table 6.2 show the results of the algorithms that participated in the 2004 ontology

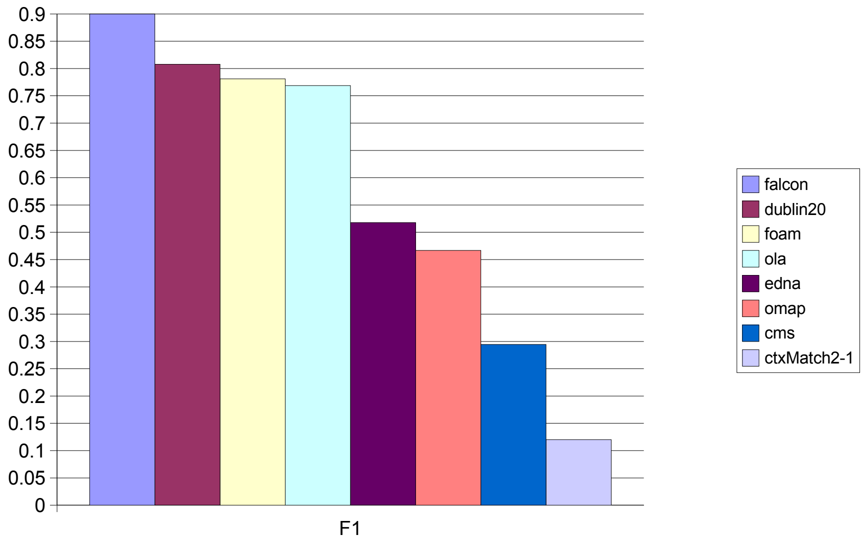


Figure 6.9: *Overall F1 of the OAEI 2005 alignments*

algo	edna		falcon		foam		ctxMatch2-1		dublin20		cms		omap		ola	
test	Prec.	Rec.	Prec.	Rec.	Prec.	Rec.	Prec.	Rec.	Prec.	Rec.	Prec.	Rec.	Prec.	Rec.	Prec.	Rec.
1xx	0.96	1.00	1.00	1.00	0.98	0.65	0.10	0.34	1.00	0.99	0.74	0.20	0.96	1.00	1.00	1.00
2xx	0.41	0.56	0.90	0.89	0.89	0.69	0.08	0.23	0.94	0.71	0.81	0.18	0.31	0.68	0.80	0.73
3xx	0.47	0.82	0.93	0.83	0.92	0.69	0.08	0.22	0.67	0.60	0.93	0.18	0.93	0.65	0.50	0.48
all	0.45	0.61	0.91	0.89	0.90	0.69	0.08	0.24	0.92	0.72	0.81	0.18	0.35	0.70	0.80	0.74

Table 6.1: *Summary of results obtained by participants of the OAEI 2005*

algo	karlsruhe2		umontreal		fujitsu		stanford	
test	Prec.	Rec.	Prec.	Rec.	Prec.	Rec.	Prec.	Rec.
1xx	NaN	0.00	0.57	0.93	0.99	1.00	0.99	1.00
2xx	0.60	0.46	0.54	0.87	0.93	0.84	0.98	0.72
3xx	0.90	0.59	0.36	0.57	0.60	0.72	0.93	0.74
all	0.65	0.40	0.52	0.83	0.88	0.85	0.98	0.77

Table 6.2: *EON 2004 results with this year's aggregation method*

algo	edna		falcon		foam		ctxMatch2-1		dublin20		cms		omap		ola	
test	Prec.	Rec.	Prec.	Rec.	Prec.	Rec.	Prec.	Rec.	Prec.	Rec.	Prec.	Rec.	Prec.	Rec.	Prec.	Rec.
1xx	0.96	1.00	1.00	1.00	0.98	0.65	0.10	0.34	1.00	0.99	0.74	0.20	0.96	1.00	1.00	1.00
2xx	0.66	0.72	0.98	0.97	0.87	0.73	0.09	0.25	0.98	0.92	0.91	0.20	0.89	0.79	0.89	0.86
3xx	0.47	0.82	0.93	0.83	0.92	0.69	0.08	0.22	0.67	0.60	0.93	0.18	0.93	0.65	0.50	0.48
all	0.66	0.78	0.97	0.96	0.74	0.59	0.09	0.26	0.94	0.88	0.65	0.18	0.90	0.81	0.85	0.83

Table 6.3: *This year's results on EON 2004 test benchmark*

alignment contest. Table 6.3 shows the results of the 2005 algorithms on the subset of the benchmark ontologies that was used in the 2004 contest.

For the directory task, the reference alignments were created automatically by the organisers in a way that guarantees that the reference alignment is correct, but there is no guarantee that the reference alignment is complete. Therefore, the organisers report only the average recall for these tasks. Figure 6.10 from (Euzenat et al. 2005) show how our algorithm compares to the other algorithms in the contest.

We could not obtain mappings for the medical ontologies. These ontologies were both too large to fit into memory and also contained some constructs that caused problems with the Jena[6] RDF access and storage class library that we used in our implementation. Only two systems from the 2005 contest, Falcon and CMS from the University of Southampton, could successfully compute alignments for these two ontologies.

6.7 Conclusion

In this chapter, we have shown a new method for ontology mapping that uses established string distance metrics and an extrinsic feature representation as known from relational learning algorithms. We treat the results of the similarity computation as a bipartite graph and use well-known algorithms from graph theory to compute an optimal one-to-one mapping. With an empirical evaluation, we have shown that our basic ideas work, and that our algorithm can compete with other approaches.

Having compared the performance of our algorithm with the competitor's performance, we believe that there is a great potential for a combination of some of our ideas with methods used by others. We ignore some valuable information that comes from the ontologies, because we do not do any logical reasoning or inference. On the other hand, some of the methods proposed here, for example the post-processing steps, could

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

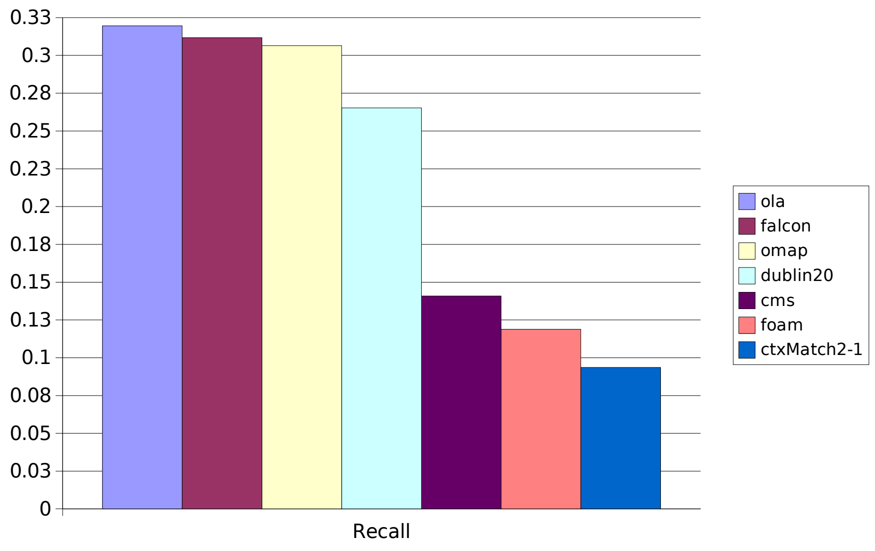


Figure 6.10: *Recall for web directories matching task*

be useful in conjunction with other base algorithms as well.

Furthermore, our algorithm could be used in a supervised way, if we use the similarity functions as nearest-neighbour classifiers or replace them with another machine learning algorithm. This is of course basically the same approach that we applied to learning Web Service annotations as described in chapter 5.

Chapter 7

Diversion: The Triskel Algorithm

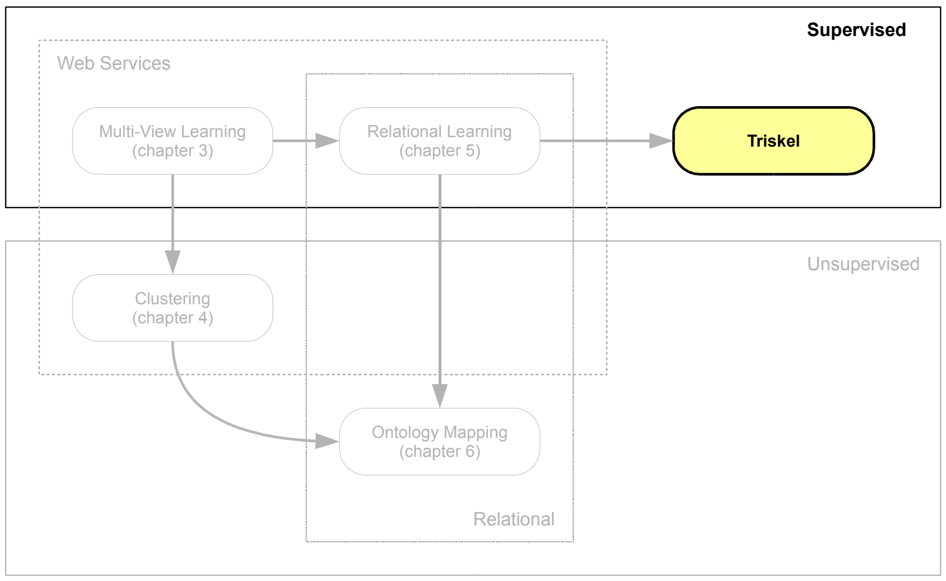

Supervised
Web Services
Multi-View Learning (chapter 3)
Relational Learning (chapter 5)
Triskel
Unsupervised
Clustering (chapter 4)
Ontology Mapping (chapter 6)
Relational

Figure 7.1: *A Celtic Triskel*

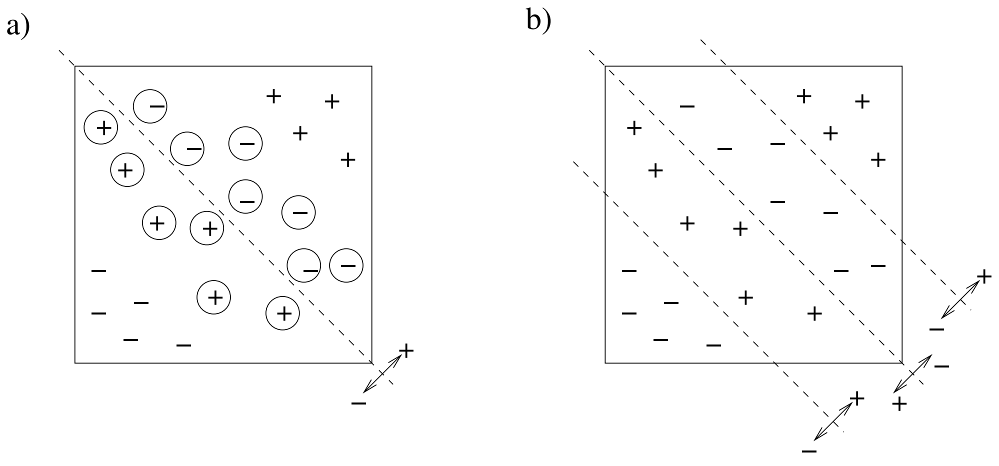


Figure 7.2: *The "layer cake" task: a) decision surface learned by a single SVM with linear kernel (circled instances are classified incorrectly); b) an ensemble of three linear SVMs that has zero training error when combined with a simple majority vote.*

7.1 Introduction

Ensemble techniques have been demonstrated to be an effective way to reduce the error of a base learner across a wide variety of tasks. The basic idea is to vote together the predictions of a set of classifiers that have been trained slightly differently for the same task. There is a strong body of theory explaining why ensemble techniques work.

Nevertheless, it is straightforward to construct learning tasks that confound existing ensemble techniques. For example, consider a synthetic"layer cake" binary learning task shown in Fig. 7.2. SVM with a linear kernel learns a decision surface with a large error. Boosting SVM does not help: at each iteration, the classifier is unable to stop making mistakes on the middle two regions; these regions then get even more weight on the next iteration, and eventually boosting gives up because it can not find a classifier with error less than 0.5.

However, Fig. 7.2(b) shows that ensemble methods are in principle well suited to this task: when combined with a simple unweighted vote, the set of three linear decision surfaces yields an ensemble that has zero error.

Motivated by this sort of learning task, we propose a novel ensemble learning algorithm called Triskel, which has two interesting features. First, Triskel learns an ensemble of classifiers that are biased to have high precision for one particular class. For example, in Fig. 7.2(b), one of the "outer" classifiers is biased to (i.e. has high precision, albeit mediocre recall, for) the positive class, and the other classifier is biased for the negative class. In contrast, most existing ensemble techniques feature ensemble members that are biased to focus on various regions of the instance space. For example, at each round, boosting focuses on instances that were classified incorrectly in previous rounds; and bagging simply involves hiding some of the training data from each ensemble member.

The second interesting feature is the manner in which Triskel assigns weights to the ensemble members. Triskel uses weighted voting like most ensemble methods, but the weights are assigned so that certain pairs of biased classifiers outweigh the rest of the ensemble, if their predictions agree. For example, in Fig. 7.2(b), the two "outer" classifiers dominate the vote if they agree, but if they disagree then the "inner" classifier casts the deciding vote. Our algorithm is named Triskel after a Celtic spiral design with three branches, see figure 7.1. In its simplest incarnation, Triskel uses an ensemble of three classifiers: one classifier biased for the positive class, one classifier biased for the negative class, and one unbiased classifier to make predictions when the others disagree.

We make the following contributions. First, we motivate and describe Triskel, our novel approach to ensemble learning, and describe various ways to construct the biased classifiers on which Triskel relies. Second, we discuss how Triskel represents a middle ground between covering and ensemble techniques such as boosting. Finally, we evaluate Triskel on a variety of real-world tasks, and demonstrate that our method often outperforms boosting, in terms of both accuracy and training time.

7.2 The Triskel Algorithm

7.2.1 Motivation

One of the problems with AdaBoost is that in each subsequent iteration the base learner is presented with more and more difficult problems. The redistribution of instance weights is based on the errors of the last learned hypothesis on the training data. Over multiple iterations, this can result in weight distributions that are too complex for the base learner to handle. For example, suppose we would like to boost a Support Vector Machine (Vapnik 1979) with a linear kernel on a synthetic data set shown in Figure 7.2a. The Figure shows the decision surface that an SVM would learn on this data in the first iteration. We can see that the distribution of errors is such that a linear decision surface will do a poor job on such a task. Specifically, the weight distribution will switch in this case between inner and outer instances after each boosting iteration without improvements to the resulting ensemble accuracy.

Nonetheless, the example in Figure 7.2a can be handled perfectly by an ensemble of three linear separators shown in Figure 7.2b combined using a majority vote. One classifier separates a part of the positive instances from the rest of positives and negatives, one classifier separates a part of the negative instances, and the remaining classifier handles the instances where the first two classifiers disagree.

An analogy between this approach and set covering can be drawn. Essentially, one classifiers covers the data instances that can be confidently classified as positive ("easy" positives), one classifier covers the data that can be confidently classified as negatives (“easy” negatives), and the last classifier is used to handle the remaining “hard” instances. Our Triskel algorithm is inspired by this idea of exploring a middle ground between ensemble and set covering methods.

In order to identify instances that can be confidently classified as positive or negative, we make use of biased classifiers. A classifier that is biased towards predicting positives will usually have a high precision on negative instances and vice versa. We train a biased classifier for each class. All instances where the biased classifiers agree are considered “easy”, all other instances are “hard”. The third classifier, the *arbiter*, is then trained only on those “hard” instances. The intuition is that the feature patterns among the “hard” instances may be different from those among the “easy” training examples. By separating away the “easy” instances and training the arbiter only on the “hard” ones, we make the learning problem for the arbiter easier since it only has to deal with a supposedly more regular subset of the data.

Like AdaBoost, we are trying to improve (boost) the classification accuracy of the base classifier on the training set by increasing the representational power using the ensemble. However, the expectation is that we can achieve better results by splitting one hard classification problem into a series of easier ones instead of progressively constructing more difficult problems as in AdaBoost.

7.2.2 The Algorithm

Consider first the Triskel algorithm for a binary classification problem. Assume that a classifier is a function mapping data instances onto a binary set of classes: $h : X \to \{-1, +1\}$. Similarly to AdaBoost, Triskel is an iterative algorithm. In each iteration, we train a pair of biased classifiers: one classifier biased towards the positive class, and one classifier biased towards the negative class. For a discussion of different ways of biasing classifiers, see section 7.2.3. Next, we evaluate the biased classifiers on the training data and obtain two sets of instances: “easy” examples, where the biased classifiers agree; and “hard” ones, where the biased classifiers disagree. To obtain the training set for the next iteration, the weights of the “easy” instances are reduced and the weights of the “hard” instances are increased. The training set obtained after the last iteration is used to train the *arbiter* classifier. Algorithm 8 shows the details.

To combine the decisions of the learned classifiers, we use a conventional weighted voting scheme, with the weights set in such a way that some ensemble members’ votes can dominate the others. Specifically, we use a sequence of exponentially decreasing weights such that if two biased classifiers from a given iteration agree on the label of a new instance, then their combined vote outweighs the votes of the classifiers from all subsequent rounds. Such weight distribution is equivalent to the simple decision rule shown in Algorithm 9.

Essentially, in each iteration we classify and separate the “easy” instances and then use the ensemble members from subsequent iterations to handle the remaining “hard” instances.

Algorithm 8 Triskel

```
/* To train on $\{\ldots, (x_i, y_i), \ldots\}$ $(y_i = \pm 1)$ */
Choose the method of weight adjustment:
$W_{easy} = 0; W_{hard} = 1$, or /* "separation" */
$W_{easy} = 1/2; W_{hard} = 2$ /* "soft covering" */
$D_0(i) = 1/N$ for each instance $i$
for $t = 1, 2, \ldots, K$ do
  $h_t^+$ = Learn with weights $D_{t-1}$, biased for class +1
  $h_t^-$ = Learn with weights $D_{t-1}$, biased for class -1
  $\alpha_t = 2^{K-t}$
  for each instance $i$ do
    $\Delta_{t,i} = \begin{cases} W_{easy}, & \textit{if } h_t^+(x_i) = h_t^-(x_i) = y_i \\ W_{hard}, & \textit{otherwise} \end{cases}$
    $D_t(i) = D_{t-1}(i) \cdot \Delta_{t,i}$ and normalise
  end for
end for
$h_{K+1}$ = Learn with weights $D_K$, unbiased
$\alpha_{K+1} = 1$
/* To classify instance $x$ */
return $y = \text{sign}\left[\sum_{t=1}^{K+1} \alpha_t h_t^*(x)\right]$, where $h_t^*(x) = h_t^+(x) + h_t^-(x)$ for $t \leq K$, and
$h_t^*(x) = h_{K+1}(x)$ for $t = K + 1$.
```

Algorithm 9 Combining classifier decisions

```
/* To classify instance $x$ */
for $t = 1, 2, \ldots, K$ do
  if $h_t^+(x) = h_t^-(x) = y$ then
    halt and return $y$
  end if
end for
return $h_{K+1}(x)$
```

There are two principle ways in which the instance weights can be adjusted during training. One way is to set the weights of the "easy" instances to zero, leaving the weights of the "hard" instances unchanged. In this case, the classifiers in each subsequent iteration are trained on a shrinking subset of the training data. This method is more similar to the set covering idea, since after each iteration (the covered) part of the training instances is completely removed from consideration. The problem with this method is that it may quickly "run out of instances". That is, the number of instances left in consideration may quickly become too small to train a sensible classifier.

Therefore, the second way to adjust the instance weights is more similar to boosting, when the weights of "easy" instances are reduced, while the weights of "hard" instances are increased. In our experiments, we increase or reduce the weights by the factor of 2 (see Algorithm 8).

7.2.3 Generating Biased Classifiers

Biasing techniques have been previously used for improving performance of neural net classifiers on imbalanced datasets (Murphey, Guo & Feldkamp 2004) and for adaptive voting in the ensembles of classifiers for incremental learning (Muhlbaier, Topalis & Polikar 2004).

Some machine learning algorithms have an inherent way of setting a bias. Bayesian classifiers, for example, output a probability distribution. The class with the highest posterior probability as calculated by the classifier is predicted. It is easy to bias a Bayesian classifier by either modifying the prior probabilities or to impose biased thresholds on the posterior probabilities. Support Vector Machines also use a confidence value threshold.

There are, however, more generic ways to bias classifiers. Resampling techniques have been used in literature to address the problem of imbalance in the training set. But resampling can of course also be used to create an imbalance, which is what we need for Triskel. Akbani et al. found in (Akbani, Kwek & Japkowicz 2004) that for imbalanced datasets undersampling the majority class to eliminate the bias leads to good performance, although some of the training examples are discarded.

In preliminary experiments, we tried over- and undersampling to create biased classifiers. We found that creating the bias through undersampling does not hurt the overall performance of Triskel, even if as little as 10% of the training instances of one class are kept. For some datasets, the performance was even slightly better than the approach with oversampling. Additionally, because we drop 90% of the instances for one class, training becomes faster. Therefore we decided to use undersampling with a 10% undersampling rate for our final experiments.

7.3 Related Work

7.3.1 Relation to Covering

There is a loose relationship between Triskel and rule covering algorithms (for example (Fürnkranz 1999)). A covering algorithm tries to identify rules with high precision that cover a large number of (ideally uniformly positive or negative) training examples. These training examples are then removed from the training set, as they are covered by the rule, and rule learning continues until all examples are covered. In Triskel, identifying easy instances using biased classifiers could be seen as covering positive and negative instances, as these instances are then removed from the training set from which the arbiter is learned.

7.3.2 Comparison with Boosting

Shapire's original boosting algorithm (Shapire 1990) uses three classifiers: The first one is trained on the original dataset, the training set for the second classifier consists

Algorithm 10 Comparison of Triskel (left) to AdaBoost (right)

```
/* To train on $\{\ldots,(x_i,y_i),\ldots\}$ */
$D_0(i) = 1/N$ for each instance $i$
for $t = 1, 2, \ldots, K$ do
  $h_t^+$ = Learn(weights $D_{t-1}$, biased +1)
  $h_t^-$ = Learn(weights $D_{t-1}$, biased -1)
  $\alpha_t = 2^{K-t}$

  for each instance $i$ do
    $\Delta_{t,i} =$
    $= \begin{cases} W_{easy} & \textit{if } h_t^+(x_i) = h_t^-(x_i) = y_i \\ W_{hard} & \textit{otherwise} \end{cases}$
    $D_t(i) = D_{t-1}(i) \cdot \Delta_{t,i}$ and normalise
  end for
end for
$h_{K+1}$ = Learn(weights $D_K$, unbiased)
$\alpha_{K+1} = 1$
/* To classify instance $x$ */
return $y = \text{sign}\left[\sum_{t=1}^{K+1} \alpha_t h_t^*(x)\right]$
```

```
/* To train on $\{\ldots,(x_i,y_i),\ldots\}$ */
$D_0(i) = 1/N$ for each instance $i$
for $t = 1, 2, \ldots, K$ do
  $h_t$ = Learn(weights $D_{t-1}$, unbiased)

  $\alpha_t = \frac{1}{2}\log\frac{1-\epsilon_t}{\epsilon_t}$,
  where $\epsilon_t = \sum_i D_{t-1}(i)[\![y_i \neq h_t(x_i)]\!]$
  for each instance $i$ do
    $\Delta_{t,i} =$
    $= \begin{cases} \epsilon_t/(1-\epsilon_t) & \textit{if } h_t(x_i) = y_i \\ 1 & \textit{otherwise} \end{cases}$
    $D_t(i) = D_{t-1}(i) \cdot \Delta_{t,i}$ and normalise
  end for
end for


/* To classify instance $x$ */
return $y = \text{sign}\left[\sum_{t=1}^{K} \alpha_t h_t(x)\right]$
```

equally of instances that were classified correctly by the first classifier and instances that were incorrectly classified. A third classifier is trained on instances where the first two classifiers disagree. The predictions are combined by voting. In our algorithm we follow up on this idea, however the way we create the first two classifiers is fundamentally different. Also unlike the original boosting, we can use multiple iterations. This results in an ensemble containing more than three classifiers similar to AdaBoost.

Both AdaBoost (Freund & Shapire 1997) and Triskel try to enhance the decision surface of the ensemble by focusing on hard instances. The main difference between the two algorithms is, however, how the hard instances are defined. In AdaBoost, the hard instances are defined as the instances where the base classifier makes mistakes. In Triskel, the hard instances are defined as the instances that cannot be classified "confidently", where we assume that we can classify an instance "confidently", if the biased classifiers agree on its label.

7.3.3 Delegating Classifiers

The method that is closest to Triskel is the very recent Delegating Classifiers algorithm by Ferri et al. (Ferri, Flach & Hernandez-Orallo 2004). The idea is that a base classifier abstains from making a prediction and delegates the decision to another classifier if the confidence in the prediction is less than a threshold. They call such a classifier "cautious", since it does not make prediction where it is too uncertain.

The "hard" instances in Triskel correspond to instances where a cautious classifier would abstain. The main difference between their work and ours is that in their algorithm the decision whether the base classifier should abstain from making a prediction or not is based only on the confidence value of the base classifier. In Triskel, we use a set of biased classifiers to make this decision. The advantage here is that we do not have to rely on the quality of the confidence value of the base classifier.

Their approach is shown to be more efficient than well-known ensemble methods like Boosting or Bagging, but is slightly less accurate. We will show in the next section that Triskel is both more efficient and more accurate than AdaBoost.

7.4 Evaluation

We evaluated Triskel on several multi-class datasets from the well-known UCI repository. Because of its very good accuracy, we chose AdaBoost as the benchmark ensemble algorithm for our experiments. We used SMO (Platt 1999) as a base classifier, again because of its good performance. However, when comparing ensemble methods, accuracy is not the only important factor. The reduced error of ensemble algorithms comes at the price of a greater computational effort. Therefore, time and memory consumption has to be compared as well. Both are usually related to the ensemble size.

Because SMO can only handle binary problems, we had to choose a mode of splitting the multi-class problems into binary classification tasks. In all but one configurations we decided to use a one-against-one scheme: A binary classifier is contructed for all pairwise combinations of two classes. This means that for a dataset with k classes it is necessary to train $\frac{k(k-1)}{2}$ classifiers. Note that on datasets with more than 3 classes, this setup is computationally more expensive than a one-against-all scheme, but generally leads to a much better performance.

In conjunction with Triskel it is possible to use a compromise between one-vs-all and one-vs-one methods. We call this extension Triskel-M. For each class, a binary problem is created in order to separate this class ('positive instances') from all others ('negative instances'). These classifiers are biased towards high precision on the positive class and used similar as in binary Triskel: If exactly one of the biased classifiers predicts positive, this prediction is returned. If more than one or none of the biased classifiers predict positive, the prediction of the arbiter is returned. The arbiter is trained in one-vs-one mode to achieve a better accuracy. In our experiments, we used Triskel-M with $W_{easy} = 0$ and 1 round (denoted as Triskel-M1).

For AdaBoost, boosting the binary classifiers individually yielded a better performance than using AdaBoost-M1 (Freund & Shapire 1997).

We used a standard SMO as baseline. We used three different AdaBoost-ensembles with 3, 10 and 50 rounds. We compared these against standard Triskel with 1 round and discarding easy instances for the arbiter ($W_{easy} = 0$) (Triskel-1) and against Triskel with weighting ($W_{easy} = 1/2; W_{hard} = 2$) with 2 and 4 rounds (denoted as Triskel-W2 and Triskel-W4). Note that for a (binary) Triskel the actual ensemble size is twice the number of rounds plus one.

We used the Weka framework (Witten & Frank 1999) to conduct our experiments. We evaluated all algorithms using 10-fold cross-validation with 10 randomized repetitions for statistical significance testing, using a corrected resampled t-test as implemented in the Weka experimenter. Detailed results can be found in Tables 7.1 and 7.2.

The experiments show that AdaBoost with 50 rounds does not improve the accuracy over AdaBoost with 10 rounds when using SMO as a base classifier. Triskel-W4 outperforms AdaBoost with 3 significant wins out of the 15 datasets used. This quality improvement comes at the price of higher training cost when compared to AdaBoost-10. However, it is still faster than AdaBoost-50. Triskel-W2 (i.e. with an ensemble size of 5 classifiers) achieves a performance that is comparable to AdaBoost-10 (2 wins, 2 losses), but is significantly faster.

As expected, the M1 setup for Triskel is both the least accurate but also the fastest ensemble method. Although the biased classifiers are only trained in a one-against-all mode, the ensemble can still sigificantly outperform the base SMO in one-against-one mode on the anneal.ORIG, hypothyroid and segment datasets. Because of its one-against-all nature, this setup of Triskel can even be faster than one-against-one SMO, especially on large datasets (here on the audiology, hypothyroid and soybean datasets), while not hurting accuracy.

Figures 7.4–7.4 illustrate the relation between training time and accuracy for the algorithms on four typical datasets. The data points on the Triskel line correspond to (from fastest to slowest) Triskel-M1, -1, -W2 and -W4, while the data points for AdaBoost show the setup for 3, 10 and 50 rounds. Note that in most cases the line for Triskel is above the AdaBoost line, indicating that Triskel offers a better trade-off between accuracy and speed. Triskel achieves greater accuracy in the same time, and the same accuracy can be reached faster. Furthermore, note that the highest accuracy for Triskel is usually above the highest accuracy for AdaBoost, indicating that, given enough time, Triskel can typically outperform any setting of AdaBoost.

Khoussainov has analysed the Triskel algorithm qualitatively in (Khoussainov, Heß & Kushmerick 2005). ROC analysis has recently become a popular technique for studying classification algorithms. Assume a binary classification problem. The ROC space has the false positive rate *FPr* on its X-axis and the true positive rate *TPr* on its Y-axis. A classifier (hypothesis) for a given dataset can be mapped onto a point in the ROC space. By changing the bias strength for each of the classifiers we can obtain a family of biased classifiers which would form *bias curves* in the ROC space.

A ROC analysis shows that Triskel performs best when there are concavities in these bias curves. Flach et al. have in (Flach & Wu 2005) shown that when the ROC curve has concavities there is room for improvement. They proposed that the decision of the classifier should in certain cases be reversed, which corresponds to a mirroring in ROC space.

Dataset	SMO vs	Triskel-M1		Ada-3 vs	Triskel-1		Ada-10 vs	Triskel-W2		Ada-50 vs	Triskel-W4	
anneal	97.46	97.90		99.02	97.85		99.13	99.07		98.99	99.39	
anneal.ORIG	87.44	95.83	◇	91.11	94.04	◇	90.04	94.21	◇	90.42	94.11	◇
audiology	80.77	80.10		81.43	79.58		81.39	80.28		81.39	80.42	
autos	71.34	70.27		74.95	71.31		76.35	76.73		75.92	76.59	
balance-scale	87.57	87.63		87.73	87.78		89.09	88.05		89.09	91.65	◇
Glass	57.36	56.94		60.66	61.95		62.23	63.40		62.33	63.41	
hypothyroid	93.58	93.87	◇	95.62	95.27		96.15	95.04	$*$	96.35	95.91	
lymphography	86.48	85.00		86.28	84.73		85.32	84.87		85.19	84.40	
primary-tumor	47.09	47.44		43.93	45.23		42.96	45.05		42.72	42.93	
segment	92.92	93.61	◇	93.66	94.31		94.11	95.08	◇	94.48	95.95	◇
soybean	93.10	92.56		93.32	92.65		93.28	92.93		93.29	92.71	
vehicle	74.08	74.15		75.30	75.40		77.53	77.39		79.21	79.48	
vowel	70.61	70.60		89.91	83.71	$*$	92.27	90.00	$*$	94.08	93.80	
waveform	86.48	86.49		86.50	86.43		86.45	86.44		86.44	86.36	
zoo	96.05	95.08		96.05	95.15		96.05	96.05		96.05	96.05	

Table 7.1: *Accuracy comparisons between SMO, AdaBoost and different settings of Triskel on multi-class problems from the UCI repository. ◇ denotes a significant increase in accuracy of Triskel over the other algorithm at the 0.05-level according to a corrected resampled t-test, $*$ denotes a significant decrease in accuracy. Note that compared to AdaBoost-10 and AdaBoost-50 Triskel achieves equal or better performance with a considerably smaller ensemble.*

Dataset	SMO vs Triskel-M1			Ada-3 vs Triskel-1			Ada-10 vs Triskel-W2			Ada-50 vs Triskel-W4		
anneal	4.45	5.66	◇	7.15	12.92	◇	13.78	25.07	◇	34.50	48.32	◇
anneal.ORIG	4.17	5.46	◇	10.79	11.64	◇	38.52	23.87	∗	95.87	46.41	∗
audiology	85.02	58.79	∗	120.83	538.19	◇	172.62	1069.06	◇	140.52	1065.85	◇
autos	5.05	8.54	◇	11.17	16.97	◇	36.24	30.37	∗	130.76	56.23	∗
balance-scale	0.99	2.36	◇	3.22	3.34	◇	12.27	5.45	∗	15.88	9.72	∗
Glass	4.73	8.05	◇	12.33	15.67	◇	39.10	26.58	∗	106.23	46.96	∗
hypothyroid	16.83	9.63	∗	38.24	12.84	∗	135.19	39.87	∗	289.24	70.09	∗
lymphography	1.95	2.56	◇	2.83	6.59	◇	1.47	10.73	◇	22.25	18.91	∗
primary-tumor	68.11	75.35	◇	145.22	280.4	◇	449.42	542.17	◇	1683.63	1204.90	∗
segment	8.06	11.43	◇	16.85	26.84	◇	56.43	52.77	∗	132.11	116.10	∗
soybean	61.49	37.30	∗	80.45	289.36	◇	174.30	604.43	◇	148.99	779.23	◇
vehicle	1.87	4.80	◇	7.28	6.90	∗	35.14	13.01	∗	90.06	25.63	∗
vowel	21.12	30.98	◇	44.11	62.61	◇	171.23	128.49	∗	676.99	272.55	∗
waveform	5.14	11.19	◇	47.36	8.30	∗	217.95	29.34	∗	361.99	78.60	∗
zoo	7.15	6.48		7.31	21.96	◇	1.85	39.91	◇	7.87	88.39	◇

Table 7.2: *Training time comparisons between SMO, AdaBoost and different settings of Triskel on multi-class problems from the UCI repository. ◇ and ∗ denote statistically significant differences as in Table 7.1. When comparing AdaBoost-10 and -50 to Triskel-W2 and -W4, Triskel usually trains faster due to the smaller ensemble. Note that Triskel-M1 is sometimes faster than SMO.*

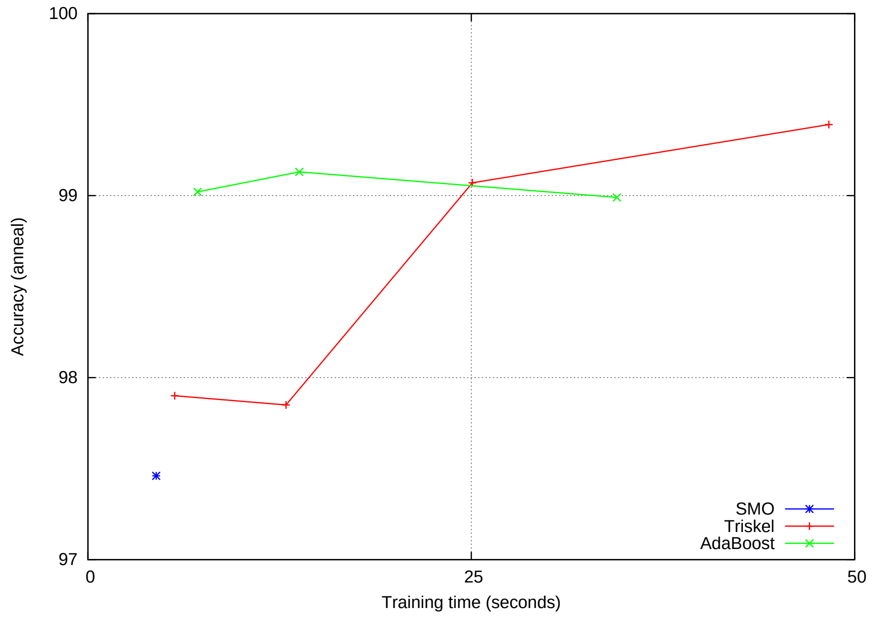


Figure 7.3: *Accuracy and training time on the "anneal" dataset*

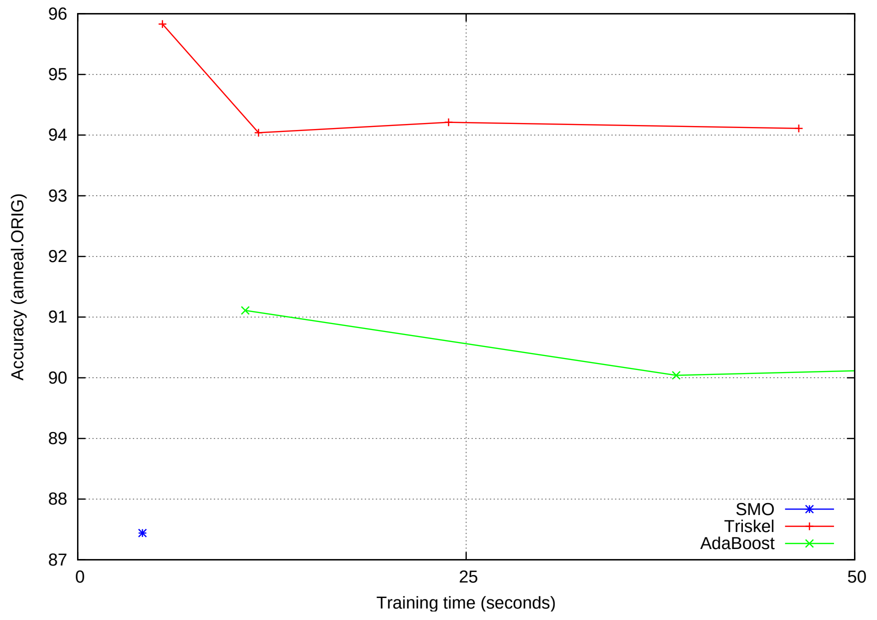


Figure 7.4: *Accuracy and training time on the "anneal.ORIG" dataset*

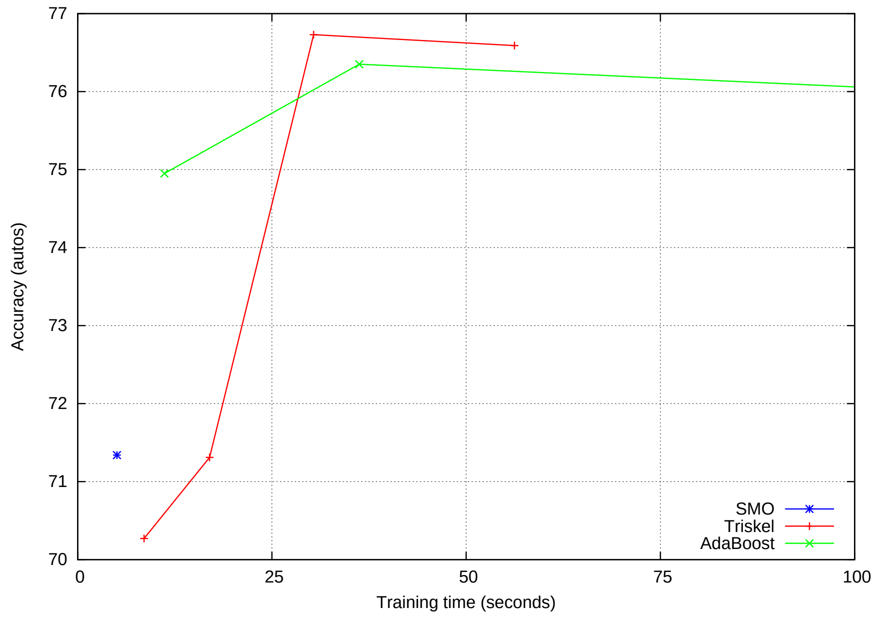


Figure 7.5: *Accuracy and training time on the "autos" dataset*

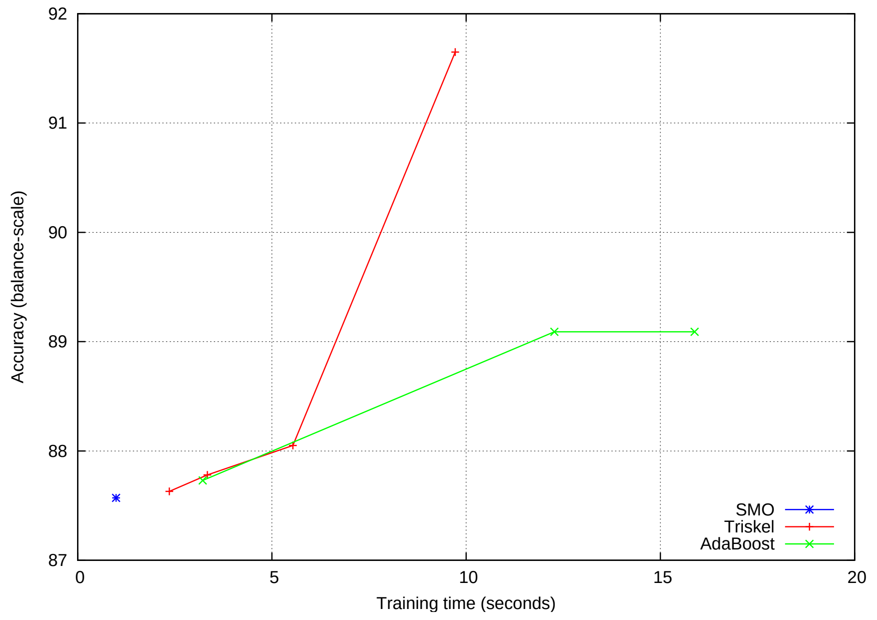


Figure 7.6: *Accuracy and training time on the "balance-scale" dataset*

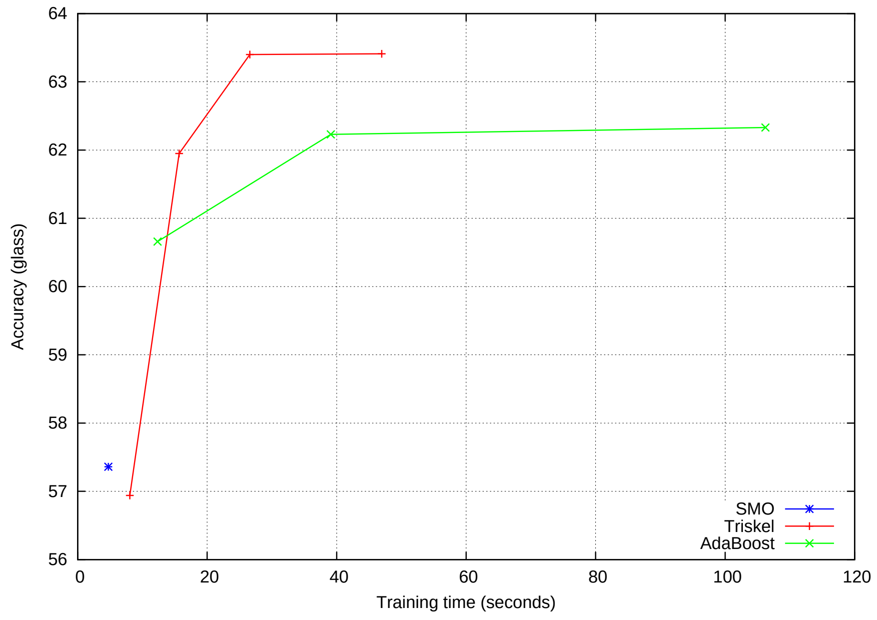


Figure 7.7: *Accuracy and training time on the "glass" dataset*

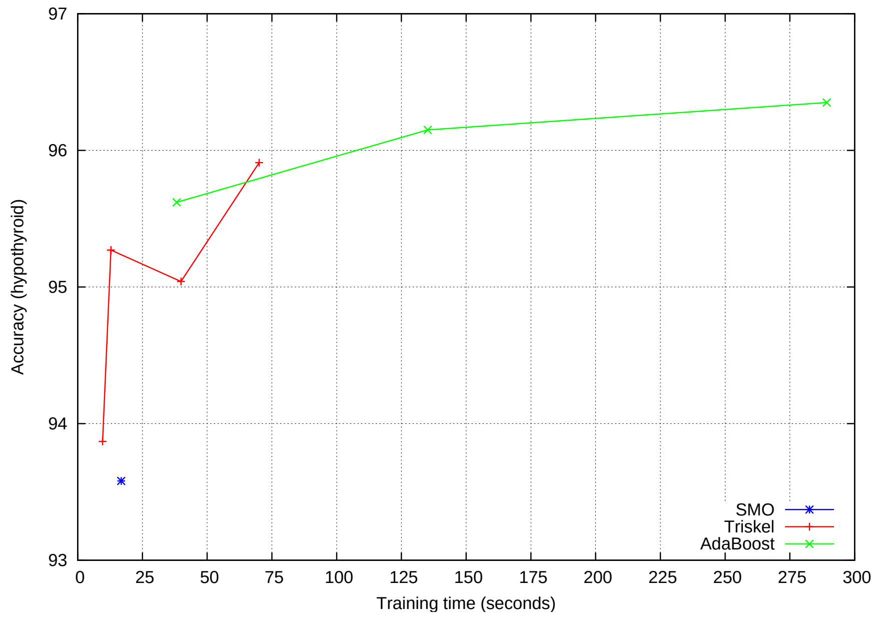


Figure 7.8: *Accuracy and training time on the "hypothyroid" dataset*

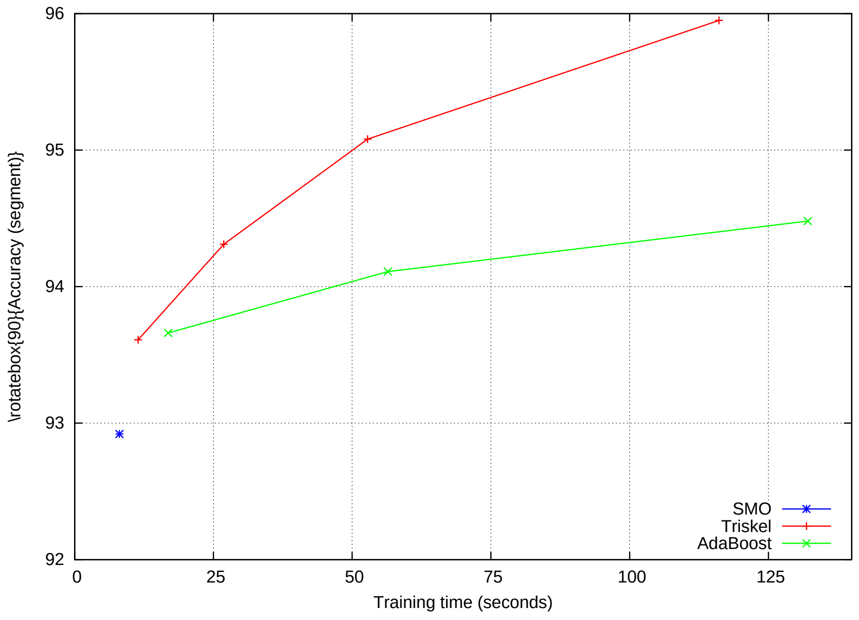


Figure 7.9: *Accuracy and training time on the "segment" dataset*

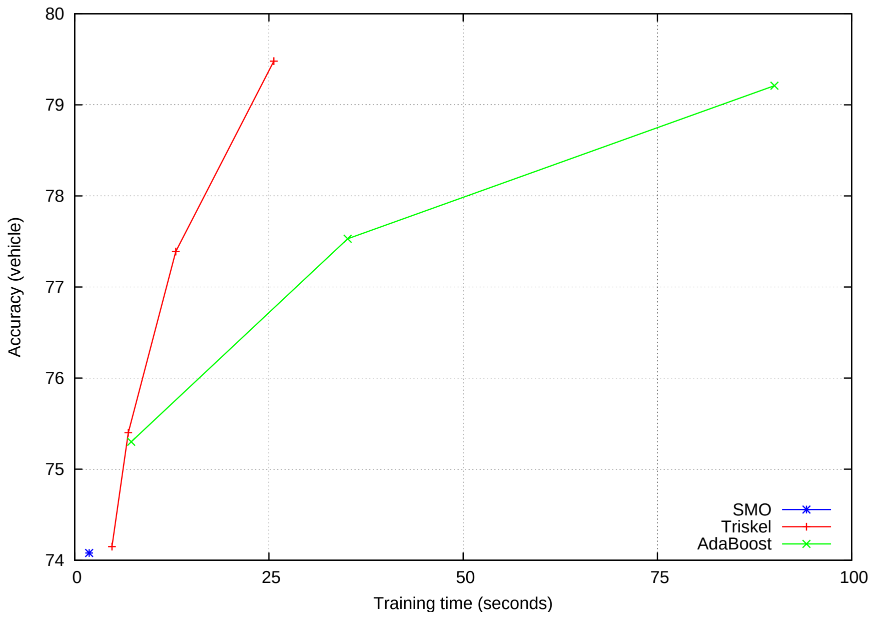


Figure 7.10: *Accuracy and training time on the "vehicle" dataset*

7.5 Future Work

7.5.1 Generating Biased Classifiers

Note that for the experiments presented in this thesis, we have used undersampling as the only method of generating the biased classifiers. While this method proved effective and has in preliminary experiments shown to achieve higher accuracy and to run faster than oversampling instances, we would like to explore other methods of generating bias in future work.

As the ROC analysis in (Khoussainov et al. 2005) shows, the effectiveness of the arbiter is limited by the quality of the biased classifiers. The biased classifiers are thus crucial for the performance of Triskel. In future work, we would like to explore the space of possible methods for generating bias, such as setting the bias of the classifier directly by means of thresholding, or to use oversampling with artificially created instances as used in the SMOTE algorithm (Chawla, Bowyer, Hall & Kegelmeyer 2002).

In preliminary experiments, we have worked on a covering-inspired way of generating the biased classifiers. To train a classifier that is biased towards high precision on positive instances, we train multiple versions of a base classifier while iteratively removing instances from the training set that the base classifier predicts as negative (i.e. instances that are covered). On classification time, an instance is predicted as positive only if all ensemble members classify it as positive. Achieving high precision on negative examples is symmetric, and the generalisation towards multi-class datasets is straightforward. Fig. 11 shows the covering-based biased classifier in pseudo-code.

Algorithm 11 Cover Negatives Algorithm

```
$D_0(i) = 1/N$ for each instance $i$
for $t = 1, 2, \ldots, K$ do
  $h_t$ = learn(instances weights $D_{t-1}$)
  for each instance $i$ do
    $D_t(i) = \begin{cases} 1 & \text{if } h_t(x_i) = 1 \\ 0 & \text{otherwise} \end{cases}$
  end for
end for
/* To classify instance $x$ */
for $t = 1, 2, \ldots, K$ do
  if $h_t(x) = -1$ then
    return $-1$
  end if
end for
return 1
```

This covering-like approach to biasing classifiers is more expressive than simple resampling approaches, because it is an ensemble itself. A Triskel classifier with covering-based biased classifiers is able to learn the correct hypothesis for a "slanted checkerboard" dataset (see Fig. 7.11), which is another example of a dataset that confounds many other algorithms. In preliminary experiments on real-world datasets, covering-like

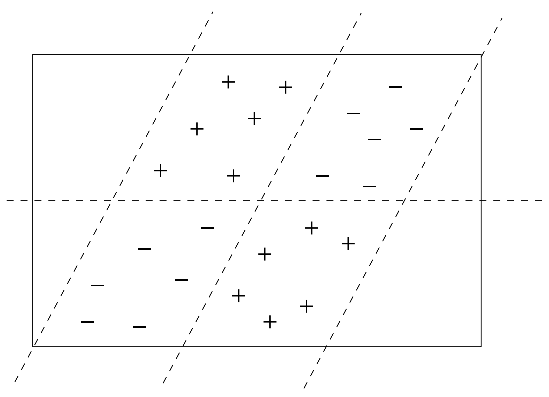


Figure 7.11: *"Slanted checkerboard" data set*

biasing improved classification accuracy slightly. On the other hand, this expressiveness is bought with the need for more ensemble members.

7.5.2 Active Learning

Active learning is a meta-learning technique that makes use of unlabelled data in a supervised way. The assumption is that the algorithm can ask an oracle (or a human operator) for the classification of an arbitrary unlabelled instance from the dataset. The goal in active learning is that by applying an effective selection strategy a learning algorithm can be trained much faster on fewer instances. This is an issue where labelled instances are expensive to obtain. Active learning algorithms try to determine instances from the unlabelled set that would improve the accuracy of the resulting classifier significantly if their label was known and then ask the oracle for the label of these instances.

In future work, we would like to examine if biased classifiers are suitable for active learning. The intuition is that knowing the label for a "hard" instance (as by the Triskel definition) should improve the accuracy of the resulting classifier more than the label of an "easy" instance, because hard instances are closer to the decision boundary. An open problem that arises here is that in datasets with concavities in the ROC curve – where Triskel in its normal form is known to perform well – the hard instances adhere to a different distribution than the easy instances, and a classifier trained on the hard instances would not generalise well on the complete dataset. Future research in that direction should address the problem how to effectively combine the features of Triskel with active learning.

7.5.3 Semi-Supervised Learning

An algorithm that makes use of unlabelled instances in a semi-supervised way is Co-Training (see section 2.4.5 in the introduction). However, Co-Training requires multiple views to be present. Given an initial set of labelled instances, classifiers trained on one view gradually label instances that are fed back as training data for classifiers for the other view.

If multiple views are not available, a variation of Co-Training that uses biased classifiers is imaginable. As in standard Co-Training, we start with an initially labelled set of instances and train biased classifiers as in Triskel. If the biased classifiers agree on a yet unlabelled instances, it is added to the training data. In future work, we would like to explore if biased classifiers can be used in such a semi-supervised setting.

7.6 Summary

We have presented a novel ensemble learning algorithm called Triskel that makes use of biased classifiers to separate "easy" and "hard" instances. In its iterative nature, it is similar in style to Boosting methods, while the way Triskel separates easy and hard instances is loosely related to covering algorithms.

Empirical results suggest that, compared to AdaBoost, Triskel offers a better tradeoff between accuracy and speed. Furthermore, the experiments show the maximum accuracy that can be achieved with Triskel is higher than the accuracy of AdaBoost.

Chapter 8

Conclusion

In this chapter, we recapitulate the research problems that have been addressed by the work in this thesis and the contribution towards their solution. In this thesis, we presented several learning approaches suited to assist developers and users of the Semantic Web, as well as an ensemble learning algorithm for general use. Before we conclude, we shortly summarise each chapter and present an outlook for future research.

8.1 Multi-View Learning for Web Services

8.1.1 Summary

In chapter 3 we presented a multi-view algorithm for classifying web services. For the web service classification task, there exist several independent sets of features to give evidence about the class of the service. For some services, plain text descriptions are available. The words used in these descriptions usually depend solely on the class of the service. Given the class, they are conditionally independent from e.g. the labels of the operations. The names of the individual parameters, for example, can also be assumed to be independent. Multi-view learning is based on the idea that the classification error for each of these views is independent from the classification error of the other views. By combining the individual predictions made by an algorithm for each view, it is possible to improve the classification accuracy. If one classifier misclassifies an instance, the intuition is that it will be outvoted by the – hopefully correct – predictions of the other classifiers. Through empirical experiments we have shown that the classification accuracy on the web services task can indeed be improved.

8.1.2 Future Directions

The experiments where web services were classified into categories were successful and encouraged us to continue our work and enhance it in order to classify not only the category of a service but also the inner parts. These experiments are summarised in section 8.3.

The insights that were gained on how to split the corpus in order to achieve a good classification performance could also be useful for other text classification problems.

8.2 Clustering Web Services

8.2.1 Summary

In chapter 5, we discussed experiments where we applied unsupervised clustering methods to web services. We evaluated several different clustering methods and compared them using several measures, among them a method based on the well-known precision and recall measures from information retrieval. We applied the precision and recall measures to pairs of instances rather than instances themselves. This allows us to use this evaluation method for clustering without the need to assign a true class from a "gold standard" to every cluster. This has the advantage that we do not penalise an algorithm overly if it computes clusters that have a different granularity than the gold standard. The experiments showed that clustering of web services into categories is possible. However, it should be obvious that we cannot expect a performance that is as good as with supervised learning.

We also introduced a variant of HAC clustering for the vector-space model that we call "Common Term", where the cluster centroid is computed not as the average of all instances but rather as the intersection. This method has the advantage that the centroids can be used as short and concise labels. It was inspired by the Word-IC algorithm (Zamir et al. 1997), but adheres to the framework of centroid-based HAC clustering.

8.2.2 Future Directions

The results we achieved in the clustering experiments are based on a single view. Very recently, multi-view clustering algorithms have been proposed (Bickel & Scheffer 2004). Using such multi-view algorithms could be an option to improve the performance on the web services task. We followed with another unsupervised method: In chapter 6 of this thesis, we discuss an algorithm that exploits the relational structure.

As with the supervised multi-view method that we discussed in chapter 3, the Common Term clustering could be also useful for general information retrieval. While this is out of the scope of this thesis, it would be interesting to explore the results of the Common Term clustering algorithm on general unstructured text.

8.3 Relational Learning for Annotating Web Services

8.3.1 Summary

In chapter 5, we presented our software ASSAM, that assists an annotator in creating semantic markup for Web Services. ASSAM uses web service descriptions in the standard WSDL format as input and can export the semantic annotations in a Semantic

Web language such as OWL-S. The key feature of ASSAM is that it suggests possible annotations to the user. ASSAM uses relational machine learning to make these recommendations.

For this application, we developed an iterative relational learning algorithm. We propose two modifications to the existing iterative relational framework: First, we propose to use separate classifiers for the intrinsic and extrinsic (relational) views and vote their predictions. Second, we propose to use specialised classifiers. These are classifiers that are trained on the intrinsic view, but only on a subset of all instances. This subset is determined by the extrinsic view.

8.3.2 Future Directions

Classifying components of web services can be seen as a specialisation of a more general data integration task. We address another instance of this problem in chapter 6.

The work presented in chapter 5 ignores the issues of composing workflows. The goal of the WS-Diamond project[1] is to add diagnosibility capabilities to web services in order to build self-healing workflows. This project will also require additional metadata that we will try to generate semi-automatically.

The work of Sabou (Sabou 2004) addresses the semi-automated creation of domain ontologies. We believe that semi-automated creation and annotation should be brought together in future work.

8.4 Unsupervised Ontology Mapping

8.4.1 Summary

Ontology mapping is an important challenge in the Semantic Web. In many domains, it is not likely that everyone will agree on a single domain ontology. To ensure interoperability in such scenarios, mapping algorithms that (semi-) automatically align two ontologies become very important.

In chapter 6, we have shown an algorithm for ontology alignment that uses string distance metrics and represents extrinsic features with a feature vector. In this algorithm, we treat the results of the similarity computation as a bipartite graph and use well-known algorithms from graph theory to compute an optimal one-to-one mapping. We have shown in an empirical evaluation that the proposed algorithms work and that the performance is comparable with other state-of-the-art algorithms.

8.4.2 Future Directions

We observed that the performance of other recent algorithms and ours are about the same when looking at the overall results. Each of the algorithms has individual strengths

[1] `http://wsdiamond.di.unito.it/`

and weaknesses with respect to the nature of the different test ontologies used. Therefore, we believe that better algorithms could be created that combine techniques from different algorithms. For example, treating the mapping task as bipartite graphs has been used in very few algorithms. Also, many matching algorithms – including ours – do not perform reasoning on the description logic level of the ontologies. Using such additional information that could be acquired through an inference engine could improve the performance.

8.5 Ensembles of Biased Classifiers: The Triskel Algorithm

8.5.1 Summary

In chapter 7, we have presented a new ensemble learning algorithm called Triskel. In this algorithm, biased classifiers are used to separate the training data into "easy" and "hard" instances. Triskel is similar to Boosting and Delegating Classifiers (see (Ferri et al. 2004)), however, the way easy and hard instances are separated is fundamentally different. Triskel uses biased classifiers to determine whether an instance can be confidently classified. The final vote is cast by an "arbiter", if a set of biased classifiers disagrees on the classification of an instance. Similar to Boosting, Triskel can be used iteratively in multiple rounds. In an empirical evaluation, we show that Triskel outperforms well-known existing ensemble methods in accuracy as well as in training speed.

8.5.2 Future Directions

There are a number of features of Triskel that we would like to explore in greater detail in future work. First, the generation of the biased classifiers is crucial for the performance of the ensemble. Better methods than random undersampling might improve the performance. Second, we would like to explore if biased classifiers are suitable for problems such as active and semi-supervised learning.

8.6 Conclusion

In this thesis, we have presented a variety of novel learning algorithms. The two leitmotifs in this work are the Semantic Web and ensembles methods. Semantic Web techniques promise to tame the information overload and aid in data integration. There are several places where machine learning can help the Semantic Web. We have covered two important aspects in this thesis: First, we have applied learning algorithms to assist an annotator in creating semantic metadata. Second, we have tackled the problem of ontology mapping. In fact, these two aspect are closely related. Annotating Semantic web services can be seen as a special form of ontology mapping, since it means mapping the web service to an ontology.

The second thread through this thesis discusses ensemble methods. We have presented applications of ensemble learning to both relational and non-relational tasks in the Semantic Web context and have shown that they usually perform better than simple approaches. As a diversion from the Semantic Web theme, we have also presented a new general-purpose ensemble learning algorithm and shown that it outperforms previously existing ensemble methods.

Appendix A

Source Code Examples

For illustration purposes, this appendix presents two typical Web Services from our dataset. We print the Web Service descriptions in WSDL format as well as semantic annotations in OWL-S that were generated by ASSAM. Note that the generated OWL-S consists of several files (see sections 1.1.4 and 5.1.2).

The first example, a service for sending faxes, has no complex types, but is very well commented. The developers made use of `documentation`-tags in various places. Because there are no complex types, there is no separate concept file for this example.

The second example, a service for calculating a distance between two given coordinates, uses some XML Schema and contains no `documentation`-tags at all.

A.1 Fax Example

A.1.1 WSDL

```
<definitions name="FaxService"
targetNamespace="http://www.OneOutBox.com/wsdl/FaxService.wsdl"
xmlns:tns="http://www.OneOutBox.com/wsdl/FaxService.wsdl"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns="http://schemas.xmlsoap.org/wsdl/">


  <documentation>
    This consolidated interface allows registered members of OneOutBox
    to easily switch between Free and Paid services with the change of a
    parameter.  This design allows easy verification of interfaces using
    the Free service without major changes when commercial coverage and
    reliability are required.  Learn more at www.OneOutBox.com
  </documentation>

  <message name="FaxRequest">
    <part name="Account" type="xsd:string">
      <documentation>
        Account identification, as established through
registration at OneOutBox.com, that uniquely identifies you
and the services available to you.
      </documentation>
    </part>
    <part name="AccessCode" type="xsd:string">
      <documentation>
        Account authentication code for security.
      </documentation>
    </part>
    <part name="Service" type="xsd:string">
      <documentation>
        Type of Fax service to be used.  Must be one of
        the following strings (without the quotes):
"free" for text, advertising-based fax.
"pro" for text, paid commercial service without advertising.
"text" alias for 'pro'
"html" for HTML formatted string, using paid commercial service.
      </documentation>
    </part>
    <part name="ToNum" type="xsd:string">
      <documentation>
        The international dialing code of the recipient
FAX machine.  e.g. USA dial 1+areacode+number.
Non-numerics within the string are ignored.
      </documentation>
    </part>
    <part name="Name" type="xsd:string">
      <documentation>
        Delivery information at the destination, such as
name and mailstop.  May include spaces (or _) and RETURN (or /)
      </documentation>
    </part>
    <part name="Text" type="xsd:string">
      <documentation>
        The contents of the FAX to be delivered.  Will
be formatted roughly 80 characters wide on the page.  No limit.
      </documentation>
    </part>
  </message>

  <message name="FaxResponse">
    <part name="return" type="xsd:string">
      <documentation>
        The return code is a tracking number that
can be used to check status and itemize usage.
      </documentation>
    </part>
  </message>

  <portType name="FaxPortType">
    <operation name="SendFax">
      <input message="tns:FaxRequest" />
      <output message="tns:FaxResponse" />
    </operation>
  </portType>

  <binding name="FaxBinding" type="tns:FaxPortType">
    <soap:binding style="rpc"
transport="http://schemas.xmlsoap.org/soap/http" />

    <operation name="SendFax">
      <soap:operation soapAction="urn:Box#SendFax" />
      <input>
```

```
        <soap:body use="encoded" namespace="urn:Box"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
      </input>
      <output>
        <soap:body use="encoded" namespace="urn:Box"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
      </output>
    </operation>
  </binding>

  <service name="FaxService">
    <documentation>
      Provides a Web Services interface to worldwide
FAX transmission services, powered by 1outbox (www.1outbox.com).
    </documentation>
    <port name="FaxPort" binding="tns:FaxBinding">
      <soap:address
location="http://www.OneOutBox.com:80/cgi-bin/soap/fax.cgi" />

    </port>
  </service>
</definitions>
```

A.1.2 OWL-S Service

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<!DOCTYPE uridef[
  <!ENTITY rdf       "http://www.w3.org/1999/02/22-rdf-syntax-ns">
  <!ENTITY rdfs      "http://www.w3.org/2000/01/rdf-schema">
  <!ENTITY xsd       "http://www.w3.org/2001/XMLSchema">
  <!ENTITY owl       "http://www.w3.org/2002/07/owl">
  <!ENTITY service   "http://www.daml.org/services/owl-s/1.0/Service.owl">
  <!ENTITY process   "http://www.daml.org/services/owl-s/1.0/Process.owl">
  <!ENTITY profile   "http://www.daml.org/services/owl-s/1.0/Profile.owl">
  <!ENTITY grounding "http://www.daml.org/services/daml-s/0.7/Grounding.daml">
  <!ENTITY profileHierarchy "http://moguntia.ucd.ie/owl/ProfileHierarchy.owl">
  <!ENTITY operations "http://moguntia.ucd.ie/owl/Operations.owl">
  <!ENTITY datatypes "http://moguntia.ucd.ie/owl/Datatypes.owl">
  <!ENTITY xsl       "http://www.w3.org/1999/XSL/Transform">
  <!ENTITY the_service  "http://moguntia.ucd.ie/1outBox_SendFax_Service.owl">
  <!ENTITY the_process  "http://moguntia.ucd.ie/1outBox_SendFax_Process.owl">
  <!ENTITY the_profile   "http://moguntia.ucd.ie/1outBox_SendFax_Profile.owl">
  <!ENTITY the_wsdl      "http://moguntia.ucd.ie/1outBox_SendFax.wsdl">
  <!ENTITY the_grounding "http://moguntia.ucd.ie/1outBox_SendFax_Grounding.owl">
  <!ENTITY the_concepts  "http://moguntia.ucd.ie/1outBox_SendFax_Concepts.owl">
  <!ENTITY DEFAULT "http://moguntia.ucd.ie/1outBox_SendFax_Service.owl">
]>

<rdf:RDF
  xmlns:rdf = "&rdf;#"
  xmlns:rdfs = "&rdfs;#"
  xmlns:xsd = "&xsd;#"
  xmlns:owl = "&owl;#"
  xmlns:service = "&service;#"
  xmlns:process = "&process;#"
  xmlns:profile = "&profile;#"
  xmlns:grounding = "&grounding;#"
  xmlns:profileHierarchy = "&profileHierarchy;#"
  xmlns:operations = "&operations;#"
  xmlns:datatypes = "&datatypes;#"
  xmlns:xsl = "&xsl;#"
  xml:base = "&DEFAULT;#"
  xmlns = "&DEFAULT;#"
>

<owl:Ontology rdf:about="">
  <owl:versionInfo>
    Generated using the ASSAM OWL export module
  </owl:versionInfo>
  <owl:imports rdf:resource="&service;"/>
  <owl:imports rdf:resource="&process;"/>
  <owl:imports rdf:resource="&profile;"/>
  <owl:imports rdf:resource="&grounding;"/>
  <owl:imports rdf:resource="&the_process;"/>
  <owl:imports rdf:resource="&the_service;"/>
  <owl:imports rdf:resource="&the_grounding;"/>
  <owl:imports rdf:resource="&the_profile;"/>
  <owl:imports rdf:resource="&profileHierarchy;"/>
  <owl:imports rdf:resource="&operations;"/>
  <owl:imports rdf:resource="&datatypes;"/>
</owl:Ontology>

<service:Service rdf:ID="Service_1outBox_SendFax">
  <service:presents rdf:resource="&the_profile;"/>
  <service:describedBy rdf:resource="&the_process;"/>
  <service:supports rdf:resource="&the_grounding;"/>
</service:Service>
</rdf:RDF>
```

A.1.3 Profile

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<!DOCTYPE uridef[
  <!ENTITY rdf       "http://www.w3.org/1999/02/22-rdf-syntax-ns">
  <!ENTITY rdfs      "http://www.w3.org/2000/01/rdf-schema">
  <!ENTITY xsd       "http://www.w3.org/2001/XMLSchema">
  <!ENTITY owl       "http://www.w3.org/2002/07/owl">
  <!ENTITY service   "http://www.daml.org/services/owl-s/1.0/Service.owl">
  <!ENTITY process   "http://www.daml.org/services/owl-s/1.0/Process.owl">
  <!ENTITY profile   "http://www.daml.org/services/owl-s/1.0/Profile.owl">
  <!ENTITY grounding "http://www.daml.org/services/daml-s/0.7/Grounding.daml">
  <!ENTITY profileHierarchy "http://moguntia.ucd.ie/owl/ProfileHierarchy.owl">
  <!ENTITY operations "http://moguntia.ucd.ie/owl/Operations.owl">
  <!ENTITY datatypes "http://moguntia.ucd.ie/owl/Datatypes.owl">
  <!ENTITY xsl       "http://www.w3.org/1999/XSL/Transform">
  <!ENTITY the_service  "http://moguntia.ucd.ie/1outBox_SendFax_Service.owl">
  <!ENTITY the_process  "http://moguntia.ucd.ie/1outBox_SendFax_Process.owl">
  <!ENTITY the_profile   "http://moguntia.ucd.ie/1outBox_SendFax_Profile.owl">
  <!ENTITY the_wsdl      "http://moguntia.ucd.ie/1outBox_SendFax.wsdl">
  <!ENTITY the_grounding "http://moguntia.ucd.ie/1outBox_SendFax_Grounding.owl">
  <!ENTITY the_concepts  "http://moguntia.ucd.ie/1outBox_SendFax_Concepts.owl">
  <!ENTITY DEFAULT   "http://moguntia.ucd.ie/1outBox_SendFax_Profile.owl">
]>

<rdf:RDF
  xmlns:rdf = "&rdf;#"
  xmlns:rdfs = "&rdfs;#"
  xmlns:xsd = "&xsd;#"
  xmlns:owl = "&owl;#"
  xmlns:service = "&service;#"
  xmlns:process = "&process;#"
  xmlns:profile = "&profile;#"
  xmlns:grounding = "&grounding;#"
  xmlns:profileHierarchy = "&profileHierarchy;#"
  xmlns:operations = "&operations;#"
  xmlns:datatypes = "&datatypes;#"
  xmlns:xsl = "&xsl;#"
  xml:base = "&DEFAULT;#"
  xmlns = "&DEFAULT;#"
>

<owl:Ontology rdf:about="">
  <owl:versionInfo>
    Generated using the ASSAM OWL export module
  </owl:versionInfo>
  <owl:imports rdf:resource="&service;"/>
  <owl:imports rdf:resource="&process;"/>
  <owl:imports rdf:resource="&profile;"/>
  <owl:imports rdf:resource="&grounding;"/>
  <owl:imports rdf:resource="&the_process;"/>
  <owl:imports rdf:resource="&the_service;"/>
  <owl:imports rdf:resource="&the_grounding;"/>
  <owl:imports rdf:resource="&the_profile;"/>
  <owl:imports rdf:resource="&profileHierarchy;"/>
  <owl:imports rdf:resource="&operations;"/>
  <owl:imports rdf:resource="&datatypes;"/>
</owl:Ontology>

<profileHierarchy:Fax rdf:ID="Profile_1outBox_SendFax">
  <service:presentedBy rdf:resource="&the_service;"/>
  <profile:has_process rdf:resource="&the_process;"/>
  <profile:serviceName>1outBox_SendFax</profile:serviceName>
  <profile:textDescription>
[B@d7a7b3
  </profile:textDescription>
<profile:hasInput rdf:resource="&the_concepts;#Account_38600"/>
<profile:hasInput rdf:resource="&the_concepts;#AccessCode_38601"/>
<profile:hasInput rdf:resource="&the_concepts;#Service_38602"/>
<profile:hasInput rdf:resource="&the_concepts;#ToNum_38603"/>
<profile:hasInput rdf:resource="&the_concepts;#Name_38604"/>
<profile:hasInput rdf:resource="&the_concepts;#Text_38605"/>
<profile:hasOutput rdf:resource="&the_concepts;#return_38607"/>
</profileHierarchy:Fax>
</rdf:RDF>
```

A.1.4 Service Model

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<!DOCTYPE uridef[
  <!ENTITY rdf       "http://www.w3.org/1999/02/22-rdf-syntax-ns">
  <!ENTITY rdfs      "http://www.w3.org/2000/01/rdf-schema">
  <!ENTITY xsd       "http://www.w3.org/2001/XMLSchema">
  <!ENTITY owl       "http://www.w3.org/2002/07/owl">
  <!ENTITY service   "http://www.daml.org/services/owl-s/1.0/Service.owl">
  <!ENTITY process   "http://www.daml.org/services/owl-s/1.0/Process.owl">
  <!ENTITY profile   "http://www.daml.org/services/owl-s/1.0/Profile.owl">
  <!ENTITY grounding "http://www.daml.org/services/daml-s/0.7/Grounding.daml">
  <!ENTITY profileHierarchy "http://moguntia.ucd.ie/owl/ProfileHierarchy.owl">
  <!ENTITY operations "http://moguntia.ucd.ie/owl/Operations.owl">
  <!ENTITY datatypes "http://moguntia.ucd.ie/owl/Datatypes.owl">
  <!ENTITY xsl       "http://www.w3.org/1999/XSL/Transform">
  <!ENTITY the_service  "http://moguntia.ucd.ie/1outBox_SendFax_Service.owl">
  <!ENTITY the_process  "http://moguntia.ucd.ie/1outBox_SendFax_Process.owl">
  <!ENTITY the_profile   "http://moguntia.ucd.ie/1outBox_SendFax_Profile.owl">
  <!ENTITY the_wsdl      "http://moguntia.ucd.ie/1outBox_SendFax.wsdl">
  <!ENTITY the_grounding "http://moguntia.ucd.ie/1outBox_SendFax_Grounding.owl">
  <!ENTITY the_concepts  "http://moguntia.ucd.ie/1outBox_SendFax_Concepts.owl">
  <!ENTITY DEFAULT   "http://moguntia.ucd.ie/1outBox_SendFax_Process.owl">
]>

<rdf:RDF
  xmlns:rdf = "&rdf;#"
  xmlns:rdfs = "&rdfs;#"
  xmlns:xsd = "&xsd;#"
  xmlns:owl = "&owl;#"
  xmlns:service = "&service;#"
  xmlns:process = "&process;#"
  xmlns:profile = "&profile;#"
  xmlns:grounding = "&grounding;#"
  xmlns:profileHierarchy = "&profileHierarchy;#"
  xmlns:operations = "&operations;#"
  xmlns:datatypes = "&datatypes;#"
  xmlns:xsl = "&xsl;#"
  xml:base = "&DEFAULT;#"
  xmlns = "&DEFAULT;#"
>

<owl:Ontology rdf:about="">
  <owl:versionInfo>
    Generated using the ASSAM OWL export module
  </owl:versionInfo>
  <owl:imports rdf:resource="&service;"/>
  <owl:imports rdf:resource="&process;"/>
  <owl:imports rdf:resource="&profile;"/>
  <owl:imports rdf:resource="&grounding;"/>
  <owl:imports rdf:resource="&the_process;"/>
  <owl:imports rdf:resource="&the_service;"/>
  <owl:imports rdf:resource="&the_grounding;"/>
  <owl:imports rdf:resource="&the_profile;"/>
  <owl:imports rdf:resource="&profileHierarchy;"/>
  <owl:imports rdf:resource="&operations;"/>
  <owl:imports rdf:resource="&datatypes;"/>
</owl:Ontology>



<process:ProcessModel rdf:ID="1outBox_SendFax">
  <process:hasProcess rdf:resource="#SendFax_38598"/>
</process:ProcessModel>



<process:AtomicProcess rdf:ID="SendFax_38598">
  <process:hasInput>
    <process:Input rdf:ID="Account_38600">
      <process:parameterType rdf:resource="&xsd;#string"/>
    </process:Input>
  </process:hasInput>
  <process:hasInput>
```

```
    <process:Input rdf:ID="AccessCode_38601">
      <process:parameterType rdf:resource="&xsd;#string"/>
    </process:Input>
  </process:hasInput>
  <process:hasInput>
    <process:Input rdf:ID="Service_38602">
      <process:parameterType rdf:resource="&xsd;#string"/>
    </process:Input>
  </process:hasInput>
  <process:hasInput>
    <process:Input rdf:ID="ToNum_38603">
      <process:parameterType rdf:resource="&xsd;#string"/>
    </process:Input>
  </process:hasInput>
  <process:hasInput>
    <process:Input rdf:ID="Name_38604">
      <process:parameterType rdf:resource="&xsd;#string"/>
    </process:Input>
  </process:hasInput>
  <process:hasInput>
    <process:Input rdf:ID="Text_38605">
      <process:parameterType rdf:resource="&xsd;#string"/>
    </process:Input>
  </process:hasInput>
  <process:hasOutput>
    <process:ConditionalOutput rdf:ID="return_38607">
      <process:parameterType rdf:resource="&xsd;#string"/>
    </process:ConditionalOutput>
  </process:hasOutput>
</process:AtomicProcess>

</rdf:RDF>
```

A.1.5 Grounding

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<!DOCTYPE uridef[
  <!ENTITY rdf       "http://www.w3.org/1999/02/22-rdf-syntax-ns">
  <!ENTITY rdfs      "http://www.w3.org/2000/01/rdf-schema">
  <!ENTITY xsd       "http://www.w3.org/2001/XMLSchema">
  <!ENTITY owl       "http://www.w3.org/2002/07/owl">
  <!ENTITY service   "http://www.daml.org/services/owl-s/1.0/Service.owl">
  <!ENTITY process   "http://www.daml.org/services/owl-s/1.0/Process.owl">
  <!ENTITY profile   "http://www.daml.org/services/owl-s/1.0/Profile.owl">
  <!ENTITY grounding "http://www.daml.org/services/daml-s/0.7/Grounding.daml">
  <!ENTITY profileHierarchy "http://moguntia.ucd.ie/owl/ProfileHierarchy.owl">
  <!ENTITY operations "http://moguntia.ucd.ie/owl/Operations.owl">
  <!ENTITY datatypes "http://moguntia.ucd.ie/owl/Datatypes.owl">
  <!ENTITY xsl       "http://www.w3.org/1999/XSL/Transform">
  <!ENTITY the_service  "http://moguntia.ucd.ie/1outBox_SendFax_Service.owl">
  <!ENTITY the_process  "http://moguntia.ucd.ie/1outBox_SendFax_Process.owl">
  <!ENTITY the_profile   "http://moguntia.ucd.ie/1outBox_SendFax_Profile.owl">
  <!ENTITY the_wsdl      "http://moguntia.ucd.ie/1outBox_SendFax.wsdl">
  <!ENTITY the_grounding "http://moguntia.ucd.ie/1outBox_SendFax_Grounding.owl">
  <!ENTITY the_concepts  "http://moguntia.ucd.ie/1outBox_SendFax_Concepts.owl">
  <!ENTITY DEFAULT "http://moguntia.ucd.ie/1outBox_SendFax_Grounding.owl">
]>

<rdf:RDF
  xmlns:rdf = "&rdf;#"
  xmlns:rdfs = "&rdfs;#"
  xmlns:xsd = "&xsd;#"
  xmlns:owl = "&owl;#"
  xmlns:service = "&service;#"
  xmlns:process = "&process;#"
  xmlns:profile = "&profile;#"
  xmlns:grounding = "&grounding;#"
  xmlns:profileHierarchy = "&profileHierarchy;#"
  xmlns:operations = "&operations;#"
  xmlns:datatypes = "&datatypes;#"
  xmlns:xsl = "&xsl;#"
  xml:base = "&DEFAULT;#"
  xmlns = "&DEFAULT;#"
>

<owl:Ontology rdf:about="">
  <owl:versionInfo>
    Generated using the ASSAM OWL export module
  </owl:versionInfo>
  <owl:imports rdf:resource="&service;"/>
  <owl:imports rdf:resource="&process;"/>
  <owl:imports rdf:resource="&profile;"/>
  <owl:imports rdf:resource="&grounding;"/>
  <owl:imports rdf:resource="&the_process;"/>
  <owl:imports rdf:resource="&the_service;"/>
  <owl:imports rdf:resource="&the_grounding;"/>
  <owl:imports rdf:resource="&the_profile;"/>
  <owl:imports rdf:resource="&profileHierarchy;"/>
  <owl:imports rdf:resource="&operations;"/>
  <owl:imports rdf:resource="&datatypes;"/>
</owl:Ontology>



<!-- Grounding Instance for the Service -->

<grounding:WsdlGrounding rdf:ID="WSDLGrounding_1outBox_SendFax">
  <service:supportedBy rdf:resource="&the_service;"/>
  <grounding:hasAtomicProcessGrounding rdf:resource="&the_process;#SendFax_38598"/>
</grounding:WsdlGrounding>



<!-- Atomic Process: SendFax_38598 ID: 38598 -->

<grounding:WsdlAtomicProcessGrounding rdf:ID="WSDLGrounding_SendFax_38598">
  <grounding:owlsProcess rdf:resource="&the_process;#SendFax_38598" />
  <grounding:wsdlOperation>
    <grounding:WsdlOperationRef>
      <grounding:portType>
        <xsd:anyURI rdf:value="&the_wsdl;#FaxPortType"/>
      </grounding:portType>
      <grounding:operation>
        <xsd:anyURI rdf:value="the_wsdl;#SendFax"/>
      </grounding:operation>
    </grounding:WsdlOperationRef>
  </grounding:wsdlOperation>
  <grounding:wsdlInputMessage>
    <xsd:anyURI rdf:value="&the_wsdl;#FaxRequest"/>
  </grounding:wsdlInputMessage>
  <grounding:wsdlInputs rdf:parseType="Collection">
    <grounding:wsdlInputMessageMap>
      <grounding:wsdlMessagePart>
        <xsd:anyURI rdf:value="&the_wsdl;#Account_38600"/>
      </grounding:wsdlMessagePart>
      <grounding:owlsParameter rdf:resource="the_process;#Account_38600"/>
    </grounding:wsdlInputMessageMap>
    <grounding:wsdlInputMessageMap>
      <grounding:wsdlMessagePart>
        <xsd:anyURI rdf:value="&the_wsdl;#AccessCode_38601"/>
      </grounding:wsdlMessagePart>
      <grounding:owlsParameter rdf:resource="the_process;#AccessCode_38601"/>
    </grounding:wsdlInputMessageMap>
    <grounding:wsdlInputMessageMap>
      <grounding:wsdlMessagePart>
        <xsd:anyURI rdf:value="&the_wsdl;#Service_38602"/>
      </grounding:wsdlMessagePart>
      <grounding:owlsParameter rdf:resource="the_process;#Service_38602"/>
    </grounding:wsdlInputMessageMap>
    <grounding:wsdlInputMessageMap>
      <grounding:wsdlMessagePart>
        <xsd:anyURI rdf:value="&the_wsdl;#ToNum_38603"/>
      </grounding:wsdlMessagePart>
      <grounding:owlsParameter rdf:resource="the_process;#ToNum_38603"/>
    </grounding:wsdlInputMessageMap>
    <grounding:wsdlInputMessageMap>
      <grounding:wsdlMessagePart>
        <xsd:anyURI rdf:value="&the_wsdl;#Name_38604"/>
      </grounding:wsdlMessagePart>
      <grounding:owlsParameter rdf:resource="the_process;#Name_38604"/>
    </grounding:wsdlInputMessageMap>
    <grounding:wsdlInputMessageMap>
      <grounding:wsdlMessagePart>
        <xsd:anyURI rdf:value="&the_wsdl;#Text_38605"/>
      </grounding:wsdlMessagePart>
      <grounding:owlsParameter rdf:resource="the_process;#Text_38605"/>
    </grounding:wsdlInputMessageMap>
  </grounding:wsdlInputs>

  <grounding:wsdlOutputMessage>
    <xsd:anyURI rdf:value="&the_wsdl;#FaxResponse_38606"/>
  </grounding:wsdlOutputMessage>
  <grounding:wsdlOutputs rdf:parseType="Collection">
    <grounding:wsdlOutputMessageMap>
      <grounding:wsdlMessagePart>
        <xsd:anyURI rdf:value="&the_wsdl;#return_38607"/>
      </grounding:wsdlMessagePart>
      <grounding:owlsParameter rdf:resource="the_process;#return_38607"/>
    </grounding:wsdlOutputMessageMap>
  </grounding:wsdlOutputs>

</grounding:WsdlAtomicProcessGrounding>

</rdf:RDF>
```

A.2 Distance Calculator Example

A.2.1 WSDL

```
<definitions xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:s0="http://innergears.com/WebServices/CalcDistance2Coords"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
targetNamespace="http://innergears.com/WebServices/CalcDistance2Coords"
xmlns="http://schemas.xmlsoap.org/wsdl/">


  <types>
    <s:schema elementFormDefault="qualified"
      targetNamespace="http://innergears.com/WebServices/CalcDistance2Coords">
      <s:element name="CalcDistTwoCoords">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="Latitude1"
               type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="Longitude1"
               type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="Latitude2"
               type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="Longitude2"
               type="s:string" />
          </s:sequence>
        </s:complexType>
      </s:element>

      <s:element name="CalcDistTwoCoordsResponse">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="1" maxOccurs="1"
             name="CalcDistTwoCoordsResult" type="s:double" />
          </s:sequence>
        </s:complexType>
      </s:element>

      <s:element name="double" type="s:double" />

    </s:schema>
  </types>

  <message name="CalcDistTwoCoordsSoapIn">
    <part name="parameters" element="s0:CalcDistTwoCoords" />
  </message>

  <message name="CalcDistTwoCoordsSoapOut">
    <part name="parameters" element="s0:CalcDistTwoCoordsResponse" />
  </message>

  <message name="CalcDistTwoCoordsHttpGetIn">
    <part name="Latitude1" type="s:string" />
    <part name="Longitude1" type="s:string" />
    <part name="Latitude2" type="s:string" />
    <part name="Longitude2" type="s:string" />
```

```
</message>

<message name="CalcDistTwoCoordsHttpGetOut">
  <part name="Body" element="s0:double" />
</message>

<message name="CalcDistTwoCoordsHttpPostIn">
  <part name="Latitude1" type="s:string" />
  <part name="Longitude1" type="s:string" />
  <part name="Latitude2" type="s:string" />
  <part name="Longitude2" type="s:string" />
</message>

<message name="CalcDistTwoCoordsHttpPostOut">
  <part name="Body" element="s0:double" />
</message>

<portType name="CalcDistance2CoordsSoap">
  <operation name="CalcDistTwoCoords">
    <input message="s0:CalcDistTwoCoordsSoapIn" />
    <output message="s0:CalcDistTwoCoordsSoapOut" />
  </operation>
</portType>

<portType name="CalcDistance2CoordsHttpGet">
  <operation name="CalcDistTwoCoords">
    <input message="s0:CalcDistTwoCoordsHttpGetIn" />
    <output message="s0:CalcDistTwoCoordsHttpGetOut" />
  </operation>
</portType>

<portType name="CalcDistance2CoordsHttpPost">
  <operation name="CalcDistTwoCoords">
    <input message="s0:CalcDistTwoCoordsHttpPostIn" />
    <output message="s0:CalcDistTwoCoordsHttpPostOut" />
  </operation>
</portType>

<binding name="CalcDistance2CoordsSoap" type="s0:CalcDistance2CoordsSoap">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
   style="document" />
  <operation name="CalcDistTwoCoords">
    <soap:operation
      soapAction="http://innergears.com/WebServices/
      CalcDistance2Coords/CalcDistTwoCoords"
      style="document" />
    <input>
      <soap:body use="literal" />
    </input>
    <output>
      <soap:body use="literal" />
    </output>
  </operation>
</binding>

<binding name="CalcDistance2CoordsHttpGet"
  type="s0:CalcDistance2CoordsHttpGet">
  <http:binding verb="GET" />
  <operation name="CalcDistTwoCoords">
```

```
      <http:operation location="/CalcDistTwoCoords" />
      <input>
        <http:urlEncoded />
      </input>
      <output>
        <mime:mimeXml part="Body" />
      </output>
    </operation>
  </binding>

  <binding name="CalcDistance2CoordsHttpPost"
    type="s0:CalcDistance2CoordsHttpPost">
    <http:binding verb="POST" />
    <operation name="CalcDistTwoCoords">
      <http:operation location="/CalcDistTwoCoords" />
      <input>
        <mime:content type="application/x-www-form-urlencoded" />
      </input>
      <output>
        <mime:mimeXml part="Body" />
      </output>
    </operation>
  </binding>

  <service name="CalcDistance2Coords">
    <port name="CalcDistance2CoordsSoap"
      binding="s0:CalcDistance2CoordsSoap">
      <soap:address
       location="http://www.innergears.com/WebServices/
       CalcDistance2Coords/CalcDistance2Coords.asmx"/>
    </port>
    <port name="CalcDistance2CoordsHttpGet"
     binding="s0:CalcDistance2CoordsHttpGet">
      <http:address
       location="http://www.innergears.com/WebServices/
       CalcDistance2Coords/CalcDistance2Coords.asmx"/>
    </port>
    <port name="CalcDistance2CoordsHttpPost"
     binding="s0:CalcDistance2CoordsHttpPost">
      <http:address
       location="http://www.innergears.com/WebServices/
       CalcDistance2Coords/CalcDistance2Coords.asmx"/>
    </port>
  </service>
</definitions>
```

A.2.2 OWL-S Service

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<!DOCTYPE uridef[
  <!ENTITY rdf       "http://www.w3.org/1999/02/22-rdf-syntax-ns">
  <!ENTITY rdfs      "http://www.w3.org/2000/01/rdf-schema">
  <!ENTITY xsd       "http://www.w3.org/2001/XMLSchema">
  <!ENTITY owl       "http://www.w3.org/2002/07/owl">
  <!ENTITY service   "http://www.daml.org/services/owl-s/1.0/Service.owl">
  <!ENTITY process   "http://www.daml.org/services/owl-s/1.0/Process.owl">
  <!ENTITY profile   "http://www.daml.org/services/owl-s/1.0/Profile.owl">
  <!ENTITY grounding "http://www.daml.org/services/daml-s/0.7/Grounding.daml">
  <!ENTITY profileHierarchy "http://moguntia.ucd.ie/owl/ProfileHierarchy.owl">
  <!ENTITY operations "http://moguntia.ucd.ie/owl/Operations.owl">
  <!ENTITY datatypes "http://moguntia.ucd.ie/owl/Datatypes.owl">
  <!ENTITY xsl       "http://www.w3.org/1999/XSL/Transform">
  <!ENTITY the_service
   "http://moguntia.ucd.ie/Distance_between_two_coordinates_Service.owl">
  <!ENTITY the_process
   "http://moguntia.ucd.ie/Distance_between_two_coordinates_Process.owl">
  <!ENTITY the_profile
   "http://moguntia.ucd.ie/Distance_between_two_coordinates_Profile.owl">
  <!ENTITY the_wsdl
   "http://moguntia.ucd.ie/Distance_between_two_coordinates.wsdl">
  <!ENTITY the_grounding
   "http://moguntia.ucd.ie/Distance_between_two_coordinates_Grounding.owl">
  <!ENTITY the_concepts
   "http://moguntia.ucd.ie/Distance_between_two_coordinates_Concepts.owl">
  <!ENTITY DEFAULT
   "http://moguntia.ucd.ie/Distance_between_two_coordinates_Service.owl">

]>

<rdf:RDF
  xmlns:rdf = "&rdf;#"
  xmlns:rdfs = "&rdfs;#"
  xmlns:xsd = "&xsd;#"
  xmlns:owl = "&owl;#"
  xmlns:service = "&service;#"
  xmlns:process = "&process;#"
  xmlns:profile = "&profile;#"
  xmlns:grounding = "&grounding;#"
  xmlns:profileHierarchy = "&profileHierarchy;#"
  xmlns:operations = "&operations;#"
  xmlns:datatypes = "&datatypes;#"
  xmlns:xsl = "&xsl;#"
  xml:base = "&DEFAULT;#"
  xmlns = "&DEFAULT;#"
>

<owl:Ontology rdf:about="">
  <owl:versionInfo>
    Generated using the ASSAM OWL export module
  </owl:versionInfo>
  <owl:imports rdf:resource="&service;"/>
  <owl:imports rdf:resource="&process;"/>
  <owl:imports rdf:resource="&profile;"/>
  <owl:imports rdf:resource="&grounding;"/>
  <owl:imports rdf:resource="&the_process;"/>
  <owl:imports rdf:resource="&the_service;"/>
  <owl:imports rdf:resource="&the_grounding;"/>
  <owl:imports rdf:resource="&the_profile;"/>
  <owl:imports rdf:resource="&profileHierarchy;"/>
  <owl:imports rdf:resource="&operations;"/>
  <owl:imports rdf:resource="&datatypes;"/>
</owl:Ontology>

<service:Service rdf:ID="Service_Distance_between_two_coordinates">
  <service:presents rdf:resource="&the_profile;"/>
  <service:describedBy rdf:resource="&the_process;"/>
  <service:supports rdf:resource="&the_grounding;"/>
</service:Service>
</rdf:RDF>
```

A.2.3 Profile

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<!DOCTYPE uridef[
  <!ENTITY rdf       "http://www.w3.org/1999/02/22-rdf-syntax-ns">
  <!ENTITY rdfs      "http://www.w3.org/2000/01/rdf-schema">
  <!ENTITY xsd       "http://www.w3.org/2001/XMLSchema">
  <!ENTITY owl       "http://www.w3.org/2002/07/owl">
  <!ENTITY service   "http://www.daml.org/services/owl-s/1.0/Service.owl">
  <!ENTITY process   "http://www.daml.org/services/owl-s/1.0/Process.owl">
  <!ENTITY profile   "http://www.daml.org/services/owl-s/1.0/Profile.owl">
  <!ENTITY grounding "http://www.daml.org/services/daml-s/0.7/Grounding.daml">
  <!ENTITY profileHierarchy "http://moguntia.ucd.ie/owl/ProfileHierarchy.owl">
  <!ENTITY operations "http://moguntia.ucd.ie/owl/Operations.owl">
  <!ENTITY datatypes "http://moguntia.ucd.ie/owl/Datatypes.owl">
  <!ENTITY xsl       "http://www.w3.org/1999/XSL/Transform">
  <!ENTITY the_service
    "http://moguntia.ucd.ie/Distance_between_two_coordinates_Service.owl">
  <!ENTITY the_process
    "http://moguntia.ucd.ie/Distance_between_two_coordinates_Process.owl">
  <!ENTITY the_profile
    "http://moguntia.ucd.ie/Distance_between_two_coordinates_Profile.owl">
  <!ENTITY the_wsdl
    "http://moguntia.ucd.ie/Distance_between_two_coordinates.wsdl">
  <!ENTITY the_grounding
    "http://moguntia.ucd.ie/Distance_between_two_coordinates_Grounding.owl">
  <!ENTITY the_concepts
    "http://moguntia.ucd.ie/Distance_between_two_coordinates_Concepts.owl">
  <!ENTITY DEFAULT
   "http://moguntia.ucd.ie/Distance_between_two_coordinates_Profile.owl">

]>

<rdf:RDF
  xmlns:rdf = "&rdf;#"
  xmlns:rdfs = "&rdfs;#"
  xmlns:xsd = "&xsd;#"
  xmlns:owl = "&owl;#"
  xmlns:service = "&service;#"
  xmlns:process = "&process;#"
  xmlns:profile = "&profile;#"
  xmlns:grounding = "&grounding;#"
  xmlns:profileHierarchy = "&profileHierarchy;#"
  xmlns:operations = "&operations;#"
  xmlns:datatypes = "&datatypes;#"
  xmlns:xsl = "&xsl;#"
  xml:base = "&DEFAULT;#"
  xmlns = "&DEFAULT;#"
>

<owl:Ontology rdf:about="">
  <owl:versionInfo>
    Generated using the ASSAM OWL export module
  </owl:versionInfo>
  <owl:imports rdf:resource="&service;"/>
  <owl:imports rdf:resource="&process;"/>
  <owl:imports rdf:resource="&profile;"/>
  <owl:imports rdf:resource="&grounding;"/>
  <owl:imports rdf:resource="&the_process;"/>
  <owl:imports rdf:resource="&the_service;"/>
  <owl:imports rdf:resource="&the_grounding;"/>
  <owl:imports rdf:resource="&the_profile;"/>
  <owl:imports rdf:resource="&profileHierarchy;"/>
  <owl:imports rdf:resource="&operations;"/>
  <owl:imports rdf:resource="&datatypes;"/>
</owl:Ontology>

<profileHierarchy:Distance_Calculator
   rdf:ID="Profile_Distance_between_two_coordinates">
  <service:presentedBy rdf:resource="&the_service;"/>
  <profile:has_process rdf:resource="&the_process;"/>
  <profile:serviceName>Distance_between_two_coordinates</profile:serviceName>
  <profile:textDescription>
[B@496fc2
  </profile:textDescription>
<profile:hasInput rdf:resource="&the_concepts;#parameters_44358"/>
<profile:hasOutput rdf:resource="&the_concepts;#parameters_44360"/>
<profile:hasInput rdf:resource="&the_concepts;#Latitude1_44364"/>
<profile:hasInput rdf:resource="&the_concepts;#Longitude1_44365"/>
<profile:hasInput rdf:resource="&the_concepts;#Latitude2_44366"/>
<profile:hasInput rdf:resource="&the_concepts;#Longitude2_44367"/>
<profile:hasOutput rdf:resource="&the_concepts;#Body_44369"/>
<profile:hasInput rdf:resource="&the_concepts;#Latitude1_44373"/>
<profile:hasInput rdf:resource="&the_concepts;#Longitude1_44374"/>
<profile:hasInput rdf:resource="&the_concepts;#Latitude2_44375"/>
<profile:hasInput rdf:resource="&the_concepts;#Longitude2_44376"/>
<profile:hasOutput rdf:resource="&the_concepts;#Body_44378"/>
</profileHierarchy:Distance_Calculator>
</rdf:RDF>
```

A.2.4 Service Model

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<!DOCTYPE uridef[
  <!ENTITY rdf       "http://www.w3.org/1999/02/22-rdf-syntax-ns">
  <!ENTITY rdfs      "http://www.w3.org/2000/01/rdf-schema">
  <!ENTITY xsd       "http://www.w3.org/2001/XMLSchema">
  <!ENTITY owl       "http://www.w3.org/2002/07/owl">
  <!ENTITY service   "http://www.daml.org/services/owl-s/1.0/Service.owl">
  <!ENTITY process   "http://www.daml.org/services/owl-s/1.0/Process.owl">
  <!ENTITY profile   "http://www.daml.org/services/owl-s/1.0/Profile.owl">
  <!ENTITY grounding "http://www.daml.org/services/daml-s/0.7/Grounding.daml">
  <!ENTITY profileHierarchy "http://moguntia.ucd.ie/owl/ProfileHierarchy.owl">
  <!ENTITY operations "http://moguntia.ucd.ie/owl/Operations.owl">
  <!ENTITY datatypes "http://moguntia.ucd.ie/owl/Datatypes.owl">
  <!ENTITY xsl       "http://www.w3.org/1999/XSL/Transform">
  <!ENTITY the_service  "http://moguntia.ucd.ie/Distance_between_two_coordinates_Service.owl">
  <!ENTITY the_process  "http://moguntia.ucd.ie/Distance_between_two_coordinates_Process.owl">
  <!ENTITY the_profile   "http://moguntia.ucd.ie/Distance_between_two_coordinates_Profile.owl">
  <!ENTITY the_wsdl      "http://moguntia.ucd.ie/Distance_between_two_coordinates.wsdl">
  <!ENTITY the_grounding "http://moguntia.ucd.ie/Distance_between_two_coordinates_Grounding.owl">
  <!ENTITY the_concepts  "http://moguntia.ucd.ie/Distance_between_two_coordinates_Concepts.owl">
  <!ENTITY DEFAULT   "http://moguntia.ucd.ie/Distance_between_two_coordinates_Process.owl">
]>

<rdf:RDF
  xmlns:rdf = "&rdf;#"
  xmlns:rdfs = "&rdfs;#"
  xmlns:xsd = "&xsd;#"
  xmlns:owl = "&owl;#"
  xmlns:service = "&service;#"
  xmlns:process = "&process;#"
  xmlns:profile = "&profile;#"
  xmlns:grounding = "&grounding;#"
  xmlns:profileHierarchy = "&profileHierarchy;#"
  xmlns:operations = "&operations;#"
  xmlns:datatypes = "&datatypes;#"
  xmlns:xsl = "&xsl;#"
  xml:base = "&DEFAULT;#"
  xmlns = "&DEFAULT;#"
>

<owl:Ontology rdf:about="">
  <owl:versionInfo>
    Generated using the ASSAM OWL export module
  </owl:versionInfo>
  <owl:imports rdf:resource="&service;"/>
  <owl:imports rdf:resource="&process;"/>
  <owl:imports rdf:resource="&profile;"/>
  <owl:imports rdf:resource="&grounding;"/>
  <owl:imports rdf:resource="&the_process;"/>
  <owl:imports rdf:resource="&the_service;"/>
  <owl:imports rdf:resource="&the_grounding;"/>
  <owl:imports rdf:resource="&the_profile;"/>
  <owl:imports rdf:resource="&profileHierarchy;"/>
  <owl:imports rdf:resource="&operations;"/>
  <owl:imports rdf:resource="&datatypes;"/>
</owl:Ontology>



<process:ProcessModel rdf:ID="Distance_between_two_coordinates">
  <process:hasProcess rdf:resource="#CalcDistTwoCoords_44356"/>
  <process:hasProcess rdf:resource="#CalcDistTwoCoords_44362"/>
  <process:hasProcess rdf:resource="#CalcDistTwoCoords_44371"/>
</process:ProcessModel>



<process:AtomicProcess rdf:ID="CalcDistTwoCoords_44356">
  <process:hasInput>
    <process:Input rdf:ID="parameters_44358">
      <process:parameterType rdf:resource="#CalcDistTwoCoords"/>
    </process:Input>
  </process:hasInput>
  <process:hasOutput>
    <process:ConditionalOutput rdf:ID="parameters_44360">
      <process:parameterType rdf:resource="#CalcDistTwoCoordsResponse"/>
    </process:ConditionalOutput>
  </process:hasOutput>
</process:AtomicProcess>



<process:AtomicProcess rdf:ID="CalcDistTwoCoords_44362">
  <process:hasInput>
    <process:Input rdf:ID="Latitude1_44364">
      <process:parameterType rdf:resource="&xsd;#string"/>
    </process:Input>
  </process:hasInput>
  <process:hasInput>
    <process:Input rdf:ID="Longitude1_44365">
      <process:parameterType rdf:resource="&xsd;#string"/>
    </process:Input>
  </process:hasInput>
  <process:hasInput>
    <process:Input rdf:ID="Latitude2_44366">
      <process:parameterType rdf:resource="&xsd;#string"/>
    </process:Input>
  </process:hasInput>
  <process:hasInput>
    <process:Input rdf:ID="Longitude2_44367">
      <process:parameterType rdf:resource="&xsd;#string"/>
    </process:Input>
  </process:hasInput>
  <process:hasOutput>
    <process:ConditionalOutput rdf:ID="Body_44369">
      <process:parameterType rdf:resource="#double"/>
    </process:ConditionalOutput>
  </process:hasOutput>
</process:AtomicProcess>



<process:AtomicProcess rdf:ID="CalcDistTwoCoords_44371">
  <process:hasInput>
    <process:Input rdf:ID="Latitude1_44373">
      <process:parameterType rdf:resource="&xsd;#string"/>
    </process:Input>
  </process:hasInput>
  <process:hasInput>
    <process:Input rdf:ID="Longitude1_44374">
      <process:parameterType rdf:resource="&xsd;#string"/>
    </process:Input>
  </process:hasInput>
  <process:hasInput>
    <process:Input rdf:ID="Latitude2_44375">
      <process:parameterType rdf:resource="&xsd;#string"/>
    </process:Input>
  </process:hasInput>
  <process:hasInput>
    <process:Input rdf:ID="Longitude2_44376">
      <process:parameterType rdf:resource="&xsd;#string"/>
    </process:Input>
  </process:hasInput>
  <process:hasOutput>
    <process:ConditionalOutput rdf:ID="Body_44378">
      <process:parameterType rdf:resource="#double"/>
    </process:ConditionalOutput>
  </process:hasOutput>
</process:AtomicProcess>

</rdf:RDF>
```

A.2.5 Grounding

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<!DOCTYPE uridef[
  <!ENTITY rdf       "http://www.w3.org/1999/02/22-rdf-syntax-ns">
  <!ENTITY rdfs      "http://www.w3.org/2000/01/rdf-schema">
  <!ENTITY xsd       "http://www.w3.org/2001/XMLSchema">
  <!ENTITY owl       "http://www.w3.org/2002/07/owl">
  <!ENTITY service   "http://www.daml.org/services/owl-s/1.0/Service.owl">
  <!ENTITY process   "http://www.daml.org/services/owl-s/1.0/Process.owl">
  <!ENTITY profile   "http://www.daml.org/services/owl-s/1.0/Profile.owl">
  <!ENTITY grounding "http://www.daml.org/services/daml-s/0.7/Grounding.daml">
  <!ENTITY profileHierarchy "http://moguntia.ucd.ie/owl/ProfileHierarchy.owl">
  <!ENTITY operations "http://moguntia.ucd.ie/owl/Operations.owl">
  <!ENTITY datatypes "http://moguntia.ucd.ie/owl/Datatypes.owl">
  <!ENTITY xsl       "http://www.w3.org/1999/XSL/Transform">
  <!ENTITY the_service
   "http://moguntia.ucd.ie/Distance_between_two_coordinates_Service.owl">
  <!ENTITY the_process
    "http://moguntia.ucd.ie/Distance_between_two_coordinates_Process.owl">
  <!ENTITY the_profile
    "http://moguntia.ucd.ie/Distance_between_two_coordinates_Profile.owl">
  <!ENTITY the_wsdl
   "http://moguntia.ucd.ie/Distance_between_two_coordinates.wsdl">
  <!ENTITY the_grounding
   "http://moguntia.ucd.ie/Distance_between_two_coordinates_Grounding.owl">
  <!ENTITY the_concepts
    "http://moguntia.ucd.ie/Distance_between_two_coordinates_Concepts.owl">
  <!ENTITY DEFAULT
    "http://moguntia.ucd.ie/Distance_between_two_coordinates_Grounding.owl">
]>

<rdf:RDF
  xmlns:rdf = "&rdf;#"
  xmlns:rdfs = "&rdfs;#"
  xmlns:xsd = "&xsd;#"
  xmlns:owl = "&owl;#"
  xmlns:service = "&service;#"
  xmlns:process = "&process;#"
  xmlns:profile = "&profile;#"
  xmlns:grounding = "&grounding;#"
  xmlns:profileHierarchy = "&profileHierarchy;#"
  xmlns:operations = "&operations;#"
  xmlns:datatypes = "&datatypes;#"
  xmlns:xsl = "&xsl;#"
  xml:base = "&DEFAULT;#"
  xmlns = "&DEFAULT;#"
>

<owl:Ontology rdf:about="">
  <owl:versionInfo>
    Generated using the ASSAM OWL export module
  </owl:versionInfo>
  <owl:imports rdf:resource="&service;"/>
  <owl:imports rdf:resource="&process;"/>
  <owl:imports rdf:resource="&profile;"/>
  <owl:imports rdf:resource="&grounding;"/>
  <owl:imports rdf:resource="&the_process;"/>
  <owl:imports rdf:resource="&the_service;"/>
  <owl:imports rdf:resource="&the_grounding;"/>
  <owl:imports rdf:resource="&the_profile;"/>
  <owl:imports rdf:resource="&profileHierarchy;"/>
  <owl:imports rdf:resource="&operations;"/>
  <owl:imports rdf:resource="&datatypes;"/>
```

```
</owl:Ontology>



<!-- Grounding Instance for the Service -->

<grounding:WsdlGrounding rdf:ID="WSDLGrounding_Distance_between_two_coordinates">
  <service:supportedBy rdf:resource="&the_service;"/>
  <grounding:hasAtomicProcessGrounding
   rdf:resource="&the_process;#CalcDistTwoCoords_44356"/>

  <grounding:hasAtomicProcessGrounding
   rdf:resource="&the_process;#CalcDistTwoCoords_44362"/>

  <grounding:hasAtomicProcessGrounding
   rdf:resource="&the_process;#CalcDistTwoCoords_44371"/>

</grounding:WsdlGrounding>



<!-- Atomic Process: CalcDistTwoCoords_44356 ID: 44356 -->

<grounding:WsdlAtomicProcessGrounding
   rdf:ID="WSDLGrounding_CalcDistTwoCoords_44356">

  <grounding:owlsProcess rdf:resource="&the_process;#CalcDistTwoCoords_44356" />
  <grounding:wsdlOperation>
    <grounding:WsdlOperationRef>
      <grounding:portType>
        <xsd:anyURI rdf:value="&the_wsdl;#CalcDistance2CoordsSoap"/>
      </grounding:portType>
      <grounding:operation>
        <xsd:anyURI rdf:value="the_wsdl;#CalcDistTwoCoords"/>
      </grounding:operation>
    </grounding:WsdlOperationRef>
  </grounding:wsdlOperation>
  <grounding:wsdlInputMessage>
    <xsd:anyURI rdf:value="&the_wsdl;#CalcDistTwoCoordsSoapIn"/>
  </grounding:wsdlInputMessage>
  <grounding:wsdlInputs rdf:parseType="Collection">
    <grounding:wsdlInputMessageMap>
      <grounding:wsdlMessagePart>
        <xsd:anyURI rdf:value="&the_wsdl;#parameters_44358"/>
      </grounding:wsdlMessagePart>

<grounding:xsltTransformation rdf:parseType="Literal">
  <xsl:stylesheet version="1.0"    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" indent="yes"/>
    <xsl:template xsl:match="/">
      <ws:CalcDistTwoCoords
       xmlns:ws="http://innergears.com/WebServices/CalcDistance2Coords"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns"
       xmlns:xsd="http://www.w3.org/2001/XMLSchema"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:the_concepts="&the_concepts;"
       xmlns:the_process="&the_process;"
       xsi:type="ws:CalcDistTwoCoords">
```

```
        <ws:Latitude1 xsi:type="xsd:string">
           <xsl:value-of
   xsl:select="rdf:RDF/the_concepts:CalcDistTwoCoords/the_concepts:Latitude1"/>
        </ws:Latitude1>

        <ws:Longitude1 xsi:type="xsd:string">
           <xsl:value-of
   xsl:select="rdf:RDF/the_concepts:CalcDistTwoCoords/the_concepts:Longitude1"/>
        </ws:Longitude1>

        <ws:Latitude2 xsi:type="xsd:string">
           <xsl:value-of
   xsl:select="rdf:RDF/the_concepts:CalcDistTwoCoords/the_concepts:Latitude2"/>
        </ws:Latitude2>

        <ws:Longitude2 xsi:type="xsd:string">
          <xsl:value-of
   xsl:select="rdf:RDF/the_concepts:CalcDistTwoCoords/the_concepts:Longitude2"/>
        </ws:Longitude2>


      </ws:CalcDistTwoCoords>
    </xsl:template>
  </xsl:stylesheet>
</grounding:xsltTransformation>
    </grounding:wsdlInputMessageMap>
  </grounding:wsdlInputs>

  <grounding:wsdlOutputMessage>
    <xsd:anyURI rdf:value="&the_wsdl;#CalcDistTwoCoordsSoapOut_44359"/>
  </grounding:wsdlOutputMessage>
  <grounding:wsdlOutputs rdf:parseType="Collection">
    <grounding:wsdlOutputMessageMap>
      <grounding:wsdlMessagePart>
        <xsd:anyURI rdf:value="&the_wsdl;#parameters_44360"/>
      </grounding:wsdlMessagePart>

<grounding:xsltTransformation rdf:parseType="Literal">
  <xsl:stylesheet version="1.0"    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" indent="yes"/>
    <xsl:template match="/">
      <rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns"
       xmlns:the_concepts="&the_concepts;"
       xmlns:the_process="&the_process;">
        <the_concepts:CalcDistTwoCoordsResponse>
        <the_concepts:CalcDistTwoCoordsResult>
          <xsl:value-of
select="parameters_44360/CalcDistTwoCoordsResult"/>
        </the_concepts:CalcDistTwoCoordsResult>
        </the_concepts:CalcDistTwoCoordsResponse>

      </rdf:RDF>
    </xsl:template>
  </xsl:stylesheet>
</grounding:xsltTransformation>
    </grounding:wsdlOutputMessageMap>
  </grounding:wsdlOutputs>
```

```
</grounding:WsdlAtomicProcessGrounding>



<!-- Atomic Process: CalcDistTwoCoords_44362 ID: 44362 -->

<grounding:WsdlAtomicProcessGrounding
  rdf:ID="WSDLGrounding_CalcDistTwoCoords_44362">

  <grounding:owlsProcess rdf:resource="&the_process;#CalcDistTwoCoords_44362" />
  <grounding:wsdlOperation>
    <grounding:WsdlOperationRef>
      <grounding:portType>
        <xsd:anyURI rdf:value="&the_wsdl;#CalcDistance2CoordsHttpPost"/>
      </grounding:portType>
      <grounding:operation>
        <xsd:anyURI rdf:value="the_wsdl;#CalcDistTwoCoords"/>
      </grounding:operation>
    </grounding:WsdlOperationRef>
  </grounding:wsdlOperation>
  <grounding:wsdlInputMessage>
    <xsd:anyURI rdf:value="&the_wsdl;#CalcDistTwoCoordsHttpPostIn"/>
  </grounding:wsdlInputMessage>
  <grounding:wsdlInputs rdf:parseType="Collection">
    <grounding:wsdlInputMessageMap>
      <grounding:wsdlMessagePart>
        <xsd:anyURI rdf:value="&the_wsdl;#Latitude1_44364"/>
      </grounding:wsdlMessagePart>
      <grounding:owlsParameter rdf:resource="the_process;#Latitude1_44364"/>
    </grounding:wsdlInputMessageMap>
    <grounding:wsdlInputMessageMap>
      <grounding:wsdlMessagePart>
        <xsd:anyURI rdf:value="&the_wsdl;#Longitude1_44365"/>
      </grounding:wsdlMessagePart>
      <grounding:owlsParameter rdf:resource="the_process;#Longitude1_44365"/>
    </grounding:wsdlInputMessageMap>
    <grounding:wsdlInputMessageMap>
      <grounding:wsdlMessagePart>
        <xsd:anyURI rdf:value="&the_wsdl;#Latitude2_44366"/>
      </grounding:wsdlMessagePart>
      <grounding:owlsParameter rdf:resource="the_process;#Latitude2_44366"/>
    </grounding:wsdlInputMessageMap>
    <grounding:wsdlInputMessageMap>
      <grounding:wsdlMessagePart>
        <xsd:anyURI rdf:value="&the_wsdl;#Longitude2_44367"/>
      </grounding:wsdlMessagePart>
      <grounding:owlsParameter rdf:resource="the_process;#Longitude2_44367"/>
    </grounding:wsdlInputMessageMap>
  </grounding:wsdlInputs>

  <grounding:wsdlOutputMessage>
    <xsd:anyURI rdf:value="&the_wsdl;#CalcDistTwoCoordsHttpPostOut_44368"/>
  </grounding:wsdlOutputMessage>
  <grounding:wsdlOutputs rdf:parseType="Collection">
    <grounding:wsdlOutputMessageMap>
      <grounding:wsdlMessagePart>
        <xsd:anyURI rdf:value="&the_wsdl;#Body_44369"/>
      </grounding:wsdlMessagePart>
```

```
<grounding:xsltTransformation rdf:parseType="Literal">
  <xsl:stylesheet version="1.0"    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" indent="yes"/>
    <xsl:template match="/">
      <rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns"
       xmlns:the_concepts="&the_concepts;"
       xmlns:the_process="&the_process;">
        <the_concepts:double>
        </the_concepts:double>

      </rdf:RDF>
    </xsl:template>
  </xsl:stylesheet>
</grounding:xsltTransformation>
    </grounding:wsdlOutputMessageMap>
  </grounding:wsdlOutputs>

</grounding:WsdlAtomicProcessGrounding>



<!-- Atomic Process: CalcDistTwoCoords_44371 ID: 44371 -->

<grounding:WsdlAtomicProcessGrounding rdf:ID="WSDLGrounding_CalcDistTwoCoords_44371">
  <grounding:owlsProcess rdf:resource="&the_process;#CalcDistTwoCoords_44371" />
  <grounding:wsdlOperation>
    <grounding:WsdlOperationRef>
      <grounding:portType>
        <xsd:anyURI rdf:value="&the_wsdl;#CalcDistance2CoordsHttpGet"/>
      </grounding:portType>
      <grounding:operation>
        <xsd:anyURI rdf:value="the_wsdl;#CalcDistTwoCoords"/>
      </grounding:operation>
    </grounding:WsdlOperationRef>
  </grounding:wsdlOperation>
  <grounding:wsdlInputMessage>
    <xsd:anyURI rdf:value="&the_wsdl;#CalcDistTwoCoordsHttpGetIn"/>
  </grounding:wsdlInputMessage>
  <grounding:wsdlInputs rdf:parseType="Collection">
    <grounding:wsdlInputMessageMap>
      <grounding:wsdlMessagePart>
        <xsd:anyURI rdf:value="&the_wsdl;#Latitude1_44373"/>
      </grounding:wsdlMessagePart>
      <grounding:owlsParameter rdf:resource="the_process;#Latitude1_44373"/>
    </grounding:wsdlInputMessageMap>
    <grounding:wsdlInputMessageMap>
      <grounding:wsdlMessagePart>
        <xsd:anyURI rdf:value="&the_wsdl;#Longitude1_44374"/>
      </grounding:wsdlMessagePart>
      <grounding:owlsParameter rdf:resource="the_process;#Longitude1_44374"/>
    </grounding:wsdlInputMessageMap>
    <grounding:wsdlInputMessageMap>
      <grounding:wsdlMessagePart>
        <xsd:anyURI rdf:value="&the_wsdl;#Latitude2_44375"/>
      </grounding:wsdlMessagePart>
      <grounding:owlsParameter rdf:resource="the_process;#Latitude2_44375"/>
    </grounding:wsdlInputMessageMap>
    <grounding:wsdlInputMessageMap>
```

```
      <grounding:wsdlMessagePart>
        <xsd:anyURI rdf:value="&the_wsdl;#Longitude2_44376"/>
      </grounding:wsdlMessagePart>
      <grounding:owlsParameter rdf:resource="the_process;#Longitude2_44376"/>
    </grounding:wsdlInputMessageMap>
  </grounding:wsdlInputs>

  <grounding:wsdlOutputMessage>
    <xsd:anyURI rdf:value="&the_wsdl;#CalcDistTwoCoordsHttpGetOut_44377"/>
  </grounding:wsdlOutputMessage>
  <grounding:wsdlOutputs rdf:parseType="Collection">
    <grounding:wsdlOutputMessageMap>
      <grounding:wsdlMessagePart>
        <xsd:anyURI rdf:value="&the_wsdl;#Body_44378"/>
      </grounding:wsdlMessagePart>

<grounding:xsltTransformation rdf:parseType="Literal">
  <xsl:stylesheet version="1.0"    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" indent="yes"/>
    <xsl:template match="/">
      <rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns"
       xmlns:the_concepts="&the_concepts;"
       xmlns:the_process="&the_process;">
        <the_concepts:double>
        </the_concepts:double>

      </rdf:RDF>
    </xsl:template>
  </xsl:stylesheet>
</grounding:xsltTransformation>
    </grounding:wsdlOutputMessageMap>
  </grounding:wsdlOutputs>

</grounding:WsdlAtomicProcessGrounding>

</rdf:RDF>
```

A.2.6 Concepts

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<!DOCTYPE uridef[
  <!ENTITY rdf       "http://www.w3.org/1999/02/22-rdf-syntax-ns">
  <!ENTITY rdfs      "http://www.w3.org/2000/01/rdf-schema">
  <!ENTITY xsd       "http://www.w3.org/2001/XMLSchema">
  <!ENTITY owl       "http://www.w3.org/2002/07/owl">
  <!ENTITY service   "http://www.daml.org/services/owl-s/1.0/Service.owl">
  <!ENTITY process   "http://www.daml.org/services/owl-s/1.0/Process.owl">
  <!ENTITY profile   "http://www.daml.org/services/owl-s/1.0/Profile.owl">
  <!ENTITY grounding "http://www.daml.org/services/daml-s/0.7/Grounding.daml">
  <!ENTITY profileHierarchy "http://moguntia.ucd.ie/owl/ProfileHierarchy.owl">
  <!ENTITY operations "http://moguntia.ucd.ie/owl/Operations.owl">
  <!ENTITY datatypes "http://moguntia.ucd.ie/owl/Datatypes.owl">
  <!ENTITY xsl       "http://www.w3.org/1999/XSL/Transform">
  <!ENTITY the_service
    "http://moguntia.ucd.ie/Distance_between_two_coordinates_Service.owl">
  <!ENTITY the_process
    "http://moguntia.ucd.ie/Distance_between_two_coordinates_Process.owl">
  <!ENTITY the_profile
    "http://moguntia.ucd.ie/Distance_between_two_coordinates_Profile.owl">
  <!ENTITY the_wsdl
    "http://moguntia.ucd.ie/Distance_between_two_coordinates.wsdl">
  <!ENTITY the_grounding
    "http://moguntia.ucd.ie/Distance_between_two_coordinates_Grounding.owl">
  <!ENTITY the_concepts
    "http://moguntia.ucd.ie/Distance_between_two_coordinates_Concepts.owl">
  <!ENTITY DEFAULT
    "http://moguntia.ucd.ie/Distance_between_two_coordinates_Concepts.owl">

]>

<rdf:RDF
  xmlns:rdf = "&rdf;#"
  xmlns:rdfs = "&rdfs;#"
  xmlns:xsd = "&xsd;#"
  xmlns:owl = "&owl;#"
  xmlns:service = "&service;#"
  xmlns:process = "&process;#"
  xmlns:profile = "&profile;#"
  xmlns:grounding = "&grounding;#"
  xmlns:profileHierarchy = "&profileHierarchy;#"
  xmlns:operations = "&operations;#"
  xmlns:datatypes = "&datatypes;#"
  xmlns:xsl = "&xsl;#"
  xml:base = "&DEFAULT;#"
  xmlns = "&DEFAULT;#"
>

<owl:Ontology rdf:about="">
  <owl:versionInfo>
    Generated using the ASSAM OWL export module
  </owl:versionInfo>
  <owl:imports rdf:resource="&service;"/>
  <owl:imports rdf:resource="&process;"/>
  <owl:imports rdf:resource="&profile;"/>
  <owl:imports rdf:resource="&grounding;"/>
  <owl:imports rdf:resource="&the_process;"/>
  <owl:imports rdf:resource="&the_service;"/>
  <owl:imports rdf:resource="&the_grounding;"/>
  <owl:imports rdf:resource="&the_profile;"/>
  <owl:imports rdf:resource="&profileHierarchy;"/>
  <owl:imports rdf:resource="&operations;"/>
  <owl:imports rdf:resource="&datatypes;"/>
</owl:Ontology>


<owl:Class rdf:ID="CalcDistTwoCoordsResponse_44347">
   <rdfs:subClassOf
rdf:resource="http://moguntia.ucd.ie/owl/Datatypes.owl#Distance_Miles"/>
</owl:Class>
<owl:DatatypeProperty rdf:ID="CalcDistTwoCoordsResult_44350">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#double"/>
  <rdfs:domain rdf:resource="#CalcDistTwoCoordsResponse_44347"/>
  <rdfs:subPropertyOf
rdf:resource="http://moguntia.ucd.ie/owl/Datatypes.owl#Distance_Miles"/>

</owl:DatatypeProperty>

<owl:Class rdf:ID="CalcDistTwoCoords_44348">
```

```
   <rdfs:subClassOf
rdf:resource="http://moguntia.ucd.ie/owl/Datatypes.owl#Coordinates"/>

</owl:Class>
<owl:DatatypeProperty rdf:ID="Latitude1_44351">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#CalcDistTwoCoords_44348"/>
  <rdfs:subPropertyOf
rdf:resource="http://moguntia.ucd.ie/owl/Datatypes.owl#Latitude"/>

</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="Longitude1_44352">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#CalcDistTwoCoords_44348"/>
  <rdfs:subPropertyOf
rdf:resource="http://moguntia.ucd.ie/owl/Datatypes.owl#Longitude"/>

</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="Latitude2_44353">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#CalcDistTwoCoords_44348"/>
  <rdfs:subPropertyOf
rdf:resource="http://moguntia.ucd.ie/owl/Datatypes.owl#Latitude"/>

</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="Longitude2_44354">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#CalcDistTwoCoords_44348"/>
  <rdfs:subPropertyOf
rdf:resource="http://moguntia.ucd.ie/owl/Datatypes.owl#Longitude"/>

</owl:DatatypeProperty>

<owl:Class rdf:ID="double_44349">
   <rdfs:subClassOf
rdf:resource="http://moguntia.ucd.ie/owl/Datatypes.owl#Distance_Miles"/>

</owl:Class>
</rdf:RDF>
```

Bibliography

Akbani, R., Kwek, S. & Japkowicz, N. (2004). Applying Support Vector Machines to Imabalanced Datasets, *in* J.-F. Boulicaut, F. Esposito, F. Giannotti & D. Pedreschi (eds), *Proceedings of the European Conference on Machine Learning*, Vol. 3201 of *Lecture Notes in Computer Science*, Springer, Pisa, Italy.

Berners-Lee, T., Hendler, J. & Lassila, O. (2001). The semantic web, *Scientific American* .

Bertsekas, D. P. & Castanon (1990). Parallel asynchronous hungarian methods for the assignment problem, *Technical Report LIDS-P-1997*.
URL: *citeseer.ist.psu.edu/bertsekas89parallel.html*

Bickel, S. & Scheffer, T. (2004). Multi-view clustering, *The Fourth IEEE International Conference on Data Mining*, Brighton, UK.

Blum, A. & Mitchell, T. (1998). Combining labeled and unlabeled data with co-training, *COLT: Proceedings of the Workshop on Computational Learning Theory*, Morgan Kaufmann.

Breiman, L. (1996). Bagging predictors, *Machine Learning* **24**(2): 123–140.

Chakrabarti, S., Dom, B. E. & Indyk, P. (1998). Enhanced hypertext categorization using hyperlinks, *in* L. M. Haas & A. Tiwary (eds), *Proceedings of SIGMOD-98, ACM International Conference on Management of Data*, ACM Press, New York, US, Seattle, US, pp. 307–318.

Chawla, N. V., Bowyer, K. W., Hall, L. O. & Kegelmeyer, W. P. (2002). SMOTE: Synthetic Minority Over-sampling TEchnique, *Journal of Artificial Intelligence Research* **16**: 341–378.

Christensen, E., Curbera, F., Meredtih, G. & Weerawarana, S. (2001). *Web Services Description Language (WSDL) 1.1*, World Wide Web Consortium.

Cohen, W. W., Ravikumar, P. & Fienberg, S. E. (2003). A comparison of string distance metrics for name-matching tasks., *Proceedings of the IJCAI-03 Workshop on Information Integration on the Web (IIWeb-03)*, pp. 73–78.

Cutting, D., Pedersen, J., Karger, D. & Tukey, J. (1992). Scatter/gather: A cluster-based approach to browsing large document collections, *Proceedings of the Fifteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 318–329.

Dean, M., Schreiber, G., Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D. L., Patel-Schneider, P. F. & Stein, L. A. (2004). Owl web ontology language reference, W3C Recommendation.
URL: *http://www.w3.org/TR/owl-ref/*

Dietterich, T. G. (2000). Ensemble methods in machine learning, *Proceedings of the First International Workshop on Multiple Classifier Systems*, Vol. 1857 of *Lecture Notes in Computer Science*, Springer, Cagliari, Italy.

Do, H. & Rahm, E. (2002). COMA - A system for flexible combination of schema matching approaches, *Proceedings of the 28th International Conference on Very Large Databases*, Hong Kong, China.

Doan, A., Domingos, P. & Halevy, A. (2003). Learning to match the schemas of data sources: A multistrategy approach, *Machine Learning* **50**(3): 279–301.

Dong, X., Havey, A., Madhavan, J., Nemes, E. & Zhang, J. (2004). Similarity search for web services, *in* M. A. Nascimento, M. T. Özsu, D. Kossmann, R. J. Miller, J. A. Blakeley & K. B. Schiefer (eds), *Proceedings of the Thirtieth International Conference on Very Large Data Bases*, Morgan Kaufmann, Toronto, Canada.

Ehrig, M. & Staab, S. (2004). QOM – quick ontology mapping, *in* S. A. McIlraith, D. Plexousakis & F. van Harmelen (eds), *Proceedings of the Third International Semantic Web Conference*, Vol. 3298 of *Lecture Notes in Computer Science*, Hiroshima, Japan, pp. 683–697.

Ehrig, M. & Sure, Y. (2004). Ontology mapping - an integrated approach, *in* C. Bussler, J. Davis, D. Fensel & R. Studer (eds), *Proceedings of the First European Semantic Web Symposium*, Vol. 3053 of *Lecture Notes in Computer Science*, Springer, Heraklion, Greece, pp. 76–91.

Euzenat, J. (2004). An API for ontology alignment, *in* S. A. McIlraith, D. Plexousakis & F. van Harmelen (eds), *Proceedings of the Third International Semantic Web Conference*, Vol. 3298 of *Lecture Notes in Computer Science*, Springer, Hiroshima, Japan.

Euzenat, J., Loup, D., Touzani, M. & Valtchev, P. (2004). Ontology alignment with OLA, *in* Y. Sure, O. Corcho, J. Euzenat & T. Hughes (eds), *Proceedings of the 3rd International Workshop on Evaluation of Ontology based Tools (EON)*, Hiroshima, Japan.

Euzenat, J., Stuckenschmidt, H. & Yatskevich, M. (2005). Introduction to the ontology alignment evaluation 2005, *K-CAP 2005 Integrating Ontologies orkshop*, Banff, Alberta, Canada.

Euzenat, J. & Valtchev, P. (2003). An integrative proximity measure for ontology alignment, *Proceedings of the ISWC-03 First International Workshop on Semantic Integration*, Sanibel Island, Florida, USA.

Ferri, C., Flach, P. & Hernandez-Orallo, J. (2004). Delegating classifiers, *21st International Conference on Machine Learning*, Banff, Alberta, Canada.

Finn, A. & Kushmerick, N. (2004). Multi-level boundary classification for information extraction, *in* J.-F. Boulicaut, F. Esposito, F. Giannotti & D. Pedreschi (eds), *Proceedings of the European Conference on Machine Learning*, Vol. 3201 of *Lecture Notes in Computer Science*, Springer, Pisa, Italy.

Flach, P. & Wu, S. (2005). Repairing concavities in ROC curves, *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, Edinburgh, Scotland, United Kingdom.

Freitag, D. & Kushmerick, N. (2000). Boosted wrapper induction, *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on on Innovative Applications of Artificial Intelligence*, Austin, Texas, USA, pp. 577–583.

Freund, Y. & Shapire, R. E. (1997). A Decision-Theoretic Generalization of On-Line Learning, *Journal of Computer and System Sciences* **55**: 119–139.

Fukuda & Matsui (1992). Finding all minimum-cost perfect matchings in bipartite graphs, *NETWORKS: Networks: An International Journal* **22**.
URL: *citeseer.ist.psu.edu/fukuda91finding.html*

Fürnkranz, J. (1999). Separate and Conquer Rule Learning, *Artificial Intelligence Review* **13**: 3–54.

Gale, D. & Shapley, L. S. (1962). College admissions and the stability of marriage, *American Mathematical Monthly* .

Galil, Z. (1986). Efficient algorithms for finding maximum matching in graphs, *Computing Surverys* **18**(1).

Gruber, T. R. (1993). Toward principles for the design of ontologies used for knowledge sharing, *in* N. Guarino (ed.), *Padua workshop on Formal Ontology.*

Hendler, J. & Lassila, O. (2001). Agents and the semantic web, *IEEE Intelligent Systems* **16**(2): 30–37.

Heß, A. & Kushmerick, N. (2003). Automatically attaching semantic metadata to web services, *Proceedings of the IJCAI-03 Workshop on Information Integration on the Web (IIWeb-03)*.

Hoshiai, T., Yamane, Y., Nakamura, D. & Tsuda, H. (2004). A semantic category matching approach to ontology alignment, *in* Y. Sure, O. Corcho, J. Euzenat & T. Hughes (eds), *Proceedings of the 3rd International Workshop on Evaluation of Ontology based Tools (EON)*, Hiroshima, Japan.

Irving, R. W., Leather, P. & Gusfield, D. (1987). An efficient algorithm for the “optimal” stable marriage, *Journal of the Association for Computing Machinery* **34**(3): 532–543.

Jaro, M. A. (1976). UNIMATCH: A record linkage system: User's manual, *Technical report*, U.S. Bureau of the Census, Washington, D.C.

Jaro, M. A. (1989). Advances in record-linkage methodology applied to matching the 1985 census of Tampa, Florida, *Journal of the American Statistical Association* **84**: 414–420.

Jeh, G. & Widom, J. (2002). Simrank: A measure of structural-context similarity, *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Edmonton, Alberta, Canada.

Joachims, T. (1999). Transductive inference for text classification using support vector machines, *Proceedings of the International Conference on Machine Learning (ICML)*.

Johnston, E. & Kushmerick, N. (2004). Aggregating web services with active invocation and ensembles of string distance metrics, *in* E. Motta, N. Shadbolt, A. Stutt & N. Gibbins (eds), *Proceedings of the 14th International Conference on Knowledge Engineering and Knowledge Management (EKAW 2004)*, Vol. 3257 of *Lecture Notes in Computer Science*, Whittlebury Hall, Northamptonshire, UK, pp. 386 – 402.

Khoussainov, R., Heß, A. & Kushmerick, N. (2005). Ensembles of biased classifiers, *Proceedings of the 22nd International Conference on Machine Learning*, Bonn, Germany.

Levenshtein, V. I. (1965). Binary codes capable of correcting deletions, insertions, and reversals, *Doklady Akademii Nauk SSSR* **163**(4): 845–848. In Russian. English Translation in Soviet Physics Doklady, 10(8) p. 707–710, 1966.

Lewis, D. D. (1991). Evaluating Text Categorization, *Proceedings of Speech and Natural Language Workshop*, Morgan Kaufmann, pp. 312–318.

Lu, Q. & Getoor, L. (2003). Link-based classification, *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, Washington, D.C., USA.

Madhavan, J., Bernstein, P. A. & Rahm, E. (2001). Generic Schema Matching with Cupid, *Proceedings of the 27th International Conference on Very Large Databases*, Rome, Italy, pp. 129–138.

Melnik, S., Molina-Garcia, H. & Rahm, E. (2002). Similarity flooding: A versatile graph matching algorithm, *Proceedings of the International Conference on Data Engineering (ICDE)*.

Mitchell, T. M. (1997). *Machine Learning*, McGraw-Hilll.

Muhlbaier, M., Topalis, A. & Polikar, R. (2004). Learn++.MT: A New Approach to Incremental Learning, *Proceedings of the Fifth International Workshop on Multiple Classifier Systems*, Vol. 3077 of *Lecture Notes in Computer Science*, Springer, Cagliari, Italy.

Munkres, J. (1957). Algorithms for the assignment and transportation problems, *SIAP* **5**(1): 32–38.

Murphey, Y. L., Guo, H. & Feldkamp, L. A. (2004). Neural learning from unbalanced data., *Applied Intelligence* **21**(2): 117–128.

Muslea, I., Minton, S. & Knoblock, C. (1999). A Hierachical Approach to Wrapper Induction, *Proceedings of the Third Internatioanl Conference Autonomous Agents*, pp. 190–197.

Neville, J. & Jensen, D. (2000). Iterative classification in relational data, *AAAI Workshop Statistical Relational Learning*.

Noy, N. F. & Musen, M. A. (2003). The PROMPT suite: interactive tools for ontology merging and mapping, *International Journal of Human-Computer Studies* **59**(6): 983–1024.

Paolucci, M., Srinivasan, N., Sycara, K. & Nishimura, T. (2003). Towards a semantic choreography of web services: From WSDL to DAML-S, *in* D. Fensel, K. Sycara & J. Mylopoulos (eds), *Proceedings of the Second International Semantic Web Conference*, Vol. 2870 of *Lecture Notes in Computer Science*, Sanibel Island, Florida, USA.

Patil, A., Oundhakar, S., Sheth, A. & Verma, K. (2004). Meteor-s web service annotation framework, *Proceedings of the 13th International World Wide Web Conference*, New York, USA.

Perkowitz, M. & Etzioni, O. (1995). Category translation: Learning to understand information on the internet, *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, Montreal, Quebec, Canada.

Platt, J. C. (1999). Fast Training of Support Vector Machines using Sequential Minimal Optimization, *in* B. Schölkopf, C. Burges & A. Smola (eds), *Advances in kernel methods: support vector learning*, MIT Press, Cambridge, MA, USA, pp. 185–208.

Popa, L., Velegrakis, Y., Miller, R. J., Hernandez, M. A. & Fagin, R. (2002). Translating web data, *Proceedings of the 28th International Conference on Very Large Databases*, Hong Kong, China, pp. 598–609.

Porter, M. F. (1980). An algorithm for suffix stripping, *Program* **14**(3): 130–137.

Sabou, M. (2004). From software APIs to web service ontologies: a semi-automatic extraction method, *in* S. A. McIlraith, D. Plexousakis & F. van Harmelen (eds), *Proceedings of the Third International Semantic Web Conference*, Vol. 3298 of *Lecture Notes in Computer Science*, Springer, Hiroshima, Japan.

Salton, G. (1989). *Automatic Text Processing*, Addison-Wesley.

Salton, G. & McGill, M. J. (1983). *Introduction to Modern Information Retrieval*, McGraw-Hill.

Schölkopf, B. (1997). *Support vector learning*, PhD thesis, GMD-Berichte No. 287, GMD-Forschungszentrum Informationstechnik, Berlin.

Shapire, R. E. (1990). The Strength of Weak Learnability, *Machine Learning* **5**(6): 197–227.

Strehl, A. (2002). *Relationship-based Clustering and Cluster Ensembles for High-dimensional Data Mining*, PhD thesis, University of Texas, Austin.

Sure, Y., Corcho, O., Euzenat, J. & Hughes, T. (eds) (2004). *Proceedings of the 3rd International Workshop on Evaluation of Ontology based Tools (EON)*, Hiroshima, Japan.

The DAML Services Coalition (2003). OWL-S 1.0, White Paper.
URL: *http://www.daml.org/services*

van Rijsbergen, C. (1979). *Information Retrieval*, 2nd edn, Butterworths, London.

Vapnik, V. N. (1979). *Vosstanovlenije Zavisimostej po Empiricheskim Dannym*, Nauka. In Russian.

Vapnik, V. N. (1982). *Estimation of Dependencies Based on Empirical Data*, Springer Verlag, New York / Berlin.

Veale, T. (1998). “soft” approaches to structural alignment in the sub-graph isomorphism problem, *IASTED International Conference on Soft Computing and Artificial Intelligence*, Cancun, Mexico.

Winkler, W. E. & Thibaudeau, Y. (1991). An application of the Fellegi-Sunter model of record linkage to the 1990 U.S. decennial census, *Technical report*, U.S. Bureau of the Census, Washington, D.C. Statistical Research Report Series RR91/09.

Witten, I. H. & Frank, E. (1999). *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*, Morgan Kaufmann, San Francisco, California, USA.

Zamir, O., Etzioni, O., Madani, O. & Karp, R. M. (1997). Fast and intuitive clustering of web documents, *Knowledge Discovery and Data Mining*, pp. 287–290.