Learning the Semantic Meaning of a Concept from the Web

By Yang Yu

Thesis submitted to the
Graduate School of the University of Maryland, Baltimore County
in partial fulfillment of the requirements for the degree of
Master of Science
2006

ABSTRACT

Title of Thesis: Learning the Semantic Meaning of a Concept from the Web

Yang Yu, Master of Science, 2006

Thesis directed by: Yun Peng

Professor

Department of Computer Science and Electrical Engineering,

University of Maryland Baltimore County

Many researchers have applied text classification techniques to the ontology mapping problem. The mapping results in these researches heavily depend on the availability of highly relevant text exemplars associated with individual concepts. However, manual preparation of exemplars is costly. In this work, we propose to automatically collect text exemplars by downloading and processing web pages listed in the search results obtained by querying a search engine. Search queries are formed for each concept according to the semantic information given in the ontology. We have implemented a prototype system and conducted a series of experiments. Given two ontologies, the process from forming search queries to calculating conditional probability of two concepts is fully automated. We assessed the effectiveness of our approach by comparing the obtained conditional probabilities in these experiments with human expectations. Our main contribution is that we explored the possibilities of utilizing web information for text classification based ontology mapping and made several valuable discoveries on its usefulness for future research.

Acknowledgments

I would like to give my sincere thanks to my thesis advisor, Dr. Yun Peng. Without his directions and encouragements, this work would not be possible.

I also would like to express my gratitude to professors in the department for their excellent teaching. Knowledge and skills I gained from Knowledge Representation, Basic Research Methods, Introduction to Artificial Intelligence and many other courses helped a lot at various steps of this research.

Finally, I would like to thank my family and friends. Their love and supports keep me going forward.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. Introduction

The semantic web is "an extension of the current web” [BLHL01], where information is marked up by ontology languages such as RDF and OWL so that it can be better processed by programs. However, it is not realistic to assume a single ontology shared by everyone. Instead, different organizations may have different ontologies for the same domain, and different ontology designers may use different terms for the same or similar concepts, reflecting their own perceptions and conceptualizations of the domain. For example, a course to teach neural networks may be called “Introduction to Neural Networks” in one university’s course ontology and be called “Introduction to Connectionist Models” in the other’s. Understanding these two courses actually teaching similar materials is not a problem to a computer science professor because in the professor’s knowledge base these two terms are very similar, i.e., they have the same or very similar meaning or semantics. However, when programs based on one ontology try to exchange information with programs based on another, problems will happen. This so-called interoperability problem has been known for a long time in software integration, and becomes more acute in the semantic web [WR04].

One of the approaches to address this interoperability problem is to map concepts defined in one ontology to semantically identical or similar concepts in another ontology. Text classification is a very powerful machine learning technique some have suggested for this purpose [DMDDH02, SPF02]. However, its success is highly dependent on the availability of text documents that are exemplars of individual concepts in the ontologies. Manually preparing a good number of highly relevant exemplars for hundreds of concepts requires great efforts and time from domain experts, greatly reducing the attractiveness of this approach, especially when dealing with large ontologies and in applications that require quick responses. We propose to automatically retrieve exemplars from the web, the largest information source available. An automated prototype system has been built based on this idea which allows us to experiment with different parameters and methods in each step of this approach. A series of experiments have shown encouraging results.

The rest of the paper is organized as follows. Section 2 provides background and motives of this work; Section 3 presents the technical details of our approach and the prototype system; Section 4 shows the experiments and results; Section 5 discusses some limitations of this approach and issues we have observed; Section 6 compares with related works; Section 7 suggests future works; and finally Section 8 gives conclusions of this work.

2. Background and Motivation

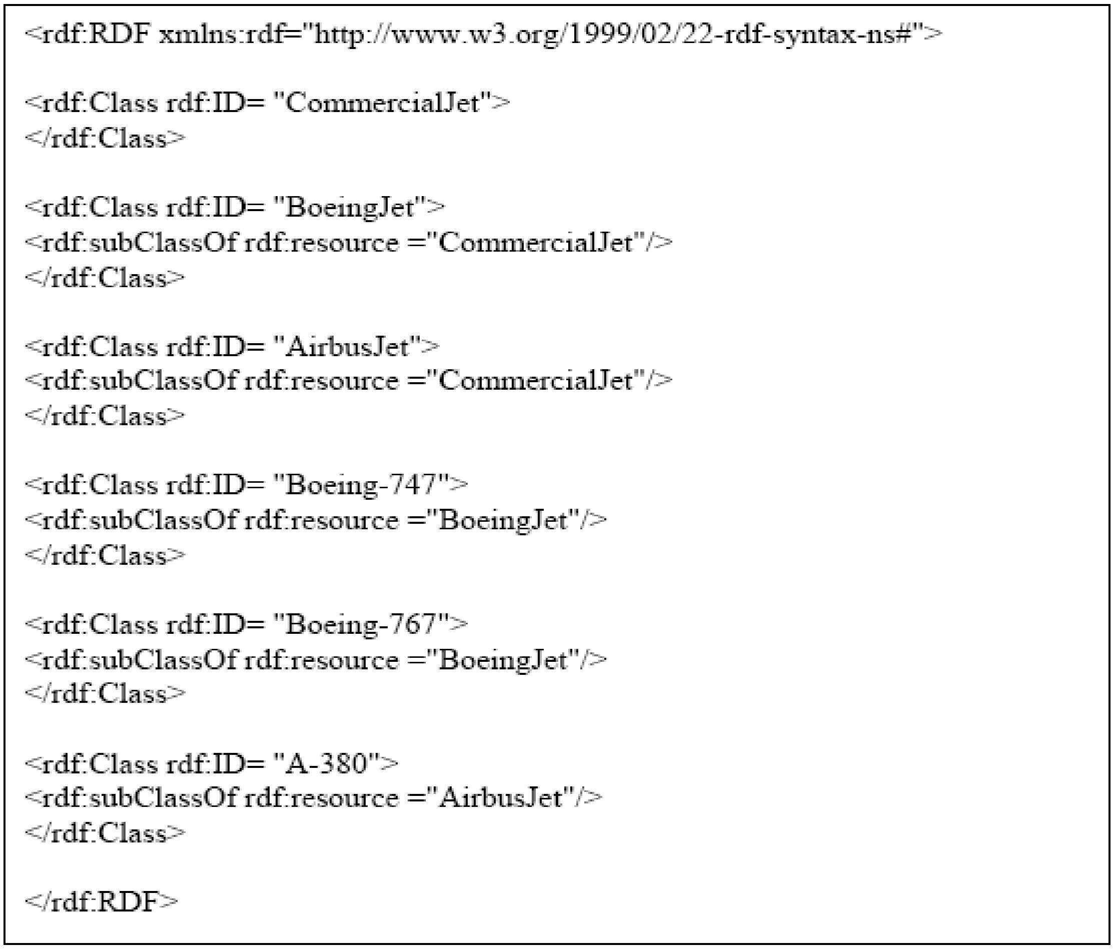
In this section, we first introduce the concepts of the semantic web and ontology. Then we briefly explain the different languages used to develop ontology files and describe several large ontologies available. After this, we define the ontology mapping problem and discuss different approaches to ontology mapping. At the end of this section, we present the hypothesis our research is set to test.

2.1 The semantic web and ontology

The Internet has enabled us to do many things more efficiently, for example, finding out the show time of a new movie being played at the theater nearest to our house, reserving a rental car at the lowest price at an airport of another city where we are going the next week, buying a book or a research paper on an interested topic that can be delivered in the shortest amount of time, and many other useful things can be done in minutes before a computer. The web has provided us enough information and applications that make all these amazing things possible. However, we, humans, are still the center of these tasks. We still need to collect, compare, and analyze most of the information and make important decisions during the process. Sometimes, the process can take a long time and we are just wondering whether programs can do more and better for us.

They cannot. The information available on the WWW is meaningful only to us humans. We understand the relation between a movie and a theater. We understand the relation between a rental car and an airport. But to a general program, these are just strings and it cannot do anything more with these strings if not specifically programmed. The solution will be that either we make a lot of complicated specialized programs, each one possessing some part of the information and being able to process them intelligently enough for some specific purposes, for example, a search engine like Google, or we can mark up the information available on the web, defining the relations between “strings”, so that every program understanding the markup language can understand the information in the same way and being able to process it more easily. We have built so many complicated programs, yet they never catch up our needs. So a universally accessible and machine-understandable web of information looks the way to go. This is the vision of the Semantic Web [BLHL01] and one way to organize and markup information is to use ontologies defined in some common languages.

In computer science, an ontology is a set of concepts each of which can have individual members, descriptions of its own properties and relations with other concepts in the set. Everybody can develop an ontology to be used by programs and other people. For example, Figure 1 shows a simple ontology defined in OWL based on [UM06]. Figure 2 shows the relations between the concepts of ontology. If a program is able to parse this markup language, then from this ontology, the program can have the knowledge that Boeing Jet is a kind of commercial jet and Boeing-747 is a kind of Boeing Jet by observing the *SubClassOf* property. If we define a "made-in" property that tells us where these commercial jets are made, the class Boeing-Jet may have such a relation with another class WA. If the ontology also has information or the program can find in some other ontologies that WA is the same as Washington State, and it is "a-part-of" USA, then these pieces of information can be easily used to answer questions like "Find all types of jets that are made in the USA".



```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">

<rdf:Class rdf:ID= "CommercialJet">
</rdf:Class>

<rdf:Class rdf:ID= "BoeingJet">
<rdf:subClassOf rdf:resource ="CommercialJet"/>
</rdf:Class>

<rdf:Class rdf:ID= "AirbusJet">
<rdf:subClassOf rdf:resource ="CommercialJet"/>
</rdf:Class>

<rdf:Class rdf:ID= "Boeing-747">
<rdf:subClassOf rdf:resource ="BoeingJet"/>
</rdf:Class>

<rdf:Class rdf:ID= "Boeing-767">
<rdf:subClassOf rdf:resource ="BoeingJet"/>
</rdf:Class>

<rdf:Class rdf:ID= "A-380">
<rdf:subClassOf rdf:resource ="AirbusJet"/>
</rdf:Class>

</rdf:RDF>
```

Figure 1 Ontology for CommercialJet in OWL

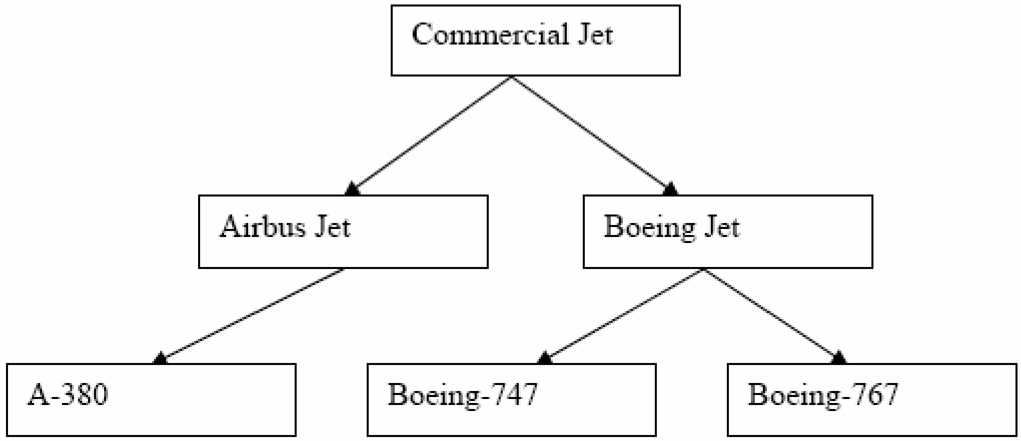


Figure 2 Ontology for CommercialJet viewed as a concept tree

By defining relations between concepts and organizing our knowledge using ontology languages, we effectively build a web of concepts, a huge integrated database, where information can be easily shared among applications and more complicated reasoning can be supported.

2.2 Ontology languages

There are quite a few languages developed to encode our knowledge into ontology files, most of which are based on XML. Here we briefly introduce some often used ones.

2.2.1 RDF and N3

RDF (Resource Description Framework) is a framework recommended by W3C to be used for metadata description and interchanging. Basically, a RDF file is formed by triples, which are statements on available resources in subject-predicate-object format. Subject is some resource, identified by a URI. Predicate is a property of the resource and object is the value of the property. N3, or notation 3 is a language designed based on RDF, which is more compact and human readable. Specifications about RDF and N3 can be found at W3C's website.

2.2.2 OWL

OWL (Web Ontology Language) is the standard language recommended by W3C to publish ontologies on the web. It is derived from DAML+OIL, an early version of ontology language based on RDF, description logics and frame languages. OWL provides a richer vocabulary, more relations between concepts and supports more complex reasoning. For example, restricted cardinality, class equality and intersection can be expressed using OWL. OWL has three variations, OWL Lite, OWL DL and OWL Full, in an increasing order of complexity. Specification of OWL can be found at the W3C website.

Ontologies used in this thesis are in OWL Lite. If relations between classes defined in an ontology file only contain subClassOf, we can view the ontology as a taxonomy and in this thesis, we sometimes refer to it as a concept tree.

2.2.3 KIF

KIF (Knowledge Interchange Format) [kif] was originally designed as an interchange format, as it is named, to be processed by computers as a representation for first-order logic. However, more and more ontology developers use it to author ontology files because its easiness to be processed. For example, the Food and Agriculture Organization of the United Nations organized its knowledge and vocabulary in several agriculture related domains into ontology files coded in several formats, one of which is KIF. A variant of KIF, SUO-KIF is used to write ontology files in SUMO [NP03]

2.3 Some large ontologies

With more and more people agreeing on the importance and inevitability of the semantic web, there are more and more projects started to develop ontologies and tools to write, publish, and use ontologies. Here we introduce a few large ontologies, some of which we will mention again in the later part of this thesis.

OpenCyc [RL02] probably is the largest human knowledge base that has been developed since the 80s by the Cycorp. The latest version of OpenCyc has 47,000 concepts forming an upper ontology covering (almost) all domains of “human consensus reality”. There are also 306,000 assertions made about these concepts. Inference engine and knowledge base browser are also provided for OpenCyc. The knowledge base is coded in a logic programming language called CycL. There are translators from CycL to Lisp and C.

WordNet [wn] is a “lexical reference system” only for English words developed at Princeton University. It is formed by synonym sets of nouns, verbs, adjective and adverbs. These synonym sets can have relations with one another. According to [NP03], “there are 66,054 noun synsets, 17,944 adjective synsets, 3,064 adverb synsets, and 12,156 verb synsets”.

SUMO (Suggested Upper Merged Ontology) [sumo] is developed by Teknowledge Corporation using SUO-KIF, a language similar to KIF, to define general-purpose concepts, which will be a foundation for more specific ontologies for different domains. According to [NP03], SUMO “has been proposed as a starter document” for IEEE’s SUO (Suggested Upper Ontology) working group. Currently it has ontologies for domains including computing services, finance, economy, terrorism, WMD, government, geography, and transportation, containing around 1,000 terms and 4,000 assertions. With SUMO, Teknowledge Corporation also provides MILO (MId Level Ontology), an ontology positioned between SUMO and detailed domain ontologies as a bridge.

2.4 Ontology mapping

Since the start of the semantic web effort, many ontologies have been developed by many organizations and individuals. For example, according to Swoogle [FD05] a semantic web search and metadata engine, as of July 2006 there are approximately 10,000 ontology documents in the web. A program that understands one ontology does not necessarily understand web pages whose semantics are specified using concepts defined in other ontologies. It is hard to centralize the ontology development effort toward a unified ontology for all people, and it is also a fact that people can use different terms for the same concept and that people can have different views on knowledge of the same domain, so different ontologies can be created for the same domain and interoperability problem can arise [WR04]. For the semantic web to work, it is imperative to relate or map concepts between such ontologies.

2.4.1 Ontology mapping definition

The problem we are trying to solve in this thesis is defined as follows:

Given an ontology Onto_A and another ontology Onto_B both defined for the same domain (or their domains overlap), we would like to be able to know the relation between any given concept C_i in Onto_A and any given concept C_j in $\text{Onto}_{B.}$ If there are *m* concepts in Onto_A and *n* concepts in Onto_B, then we would like to find a function *f*, which returns a relation *r*,

$r = f\,(Ci, Cj)$ where $i=1, \ldots, n$ and $j=1, \ldots, m$;

r = {*equivalent, subClassOf, superClassof, complement, overlapped, other*}

Completely and neatly solve the ontology mapping problem defined above is difficult. In most cases, we just want to know if C_j in Onto_B has any equivalent corresponding class C_i defined in $\text{Onto}_{A.}$ This is the easiest place to start solving the whole problem, and also where most researches, including this one, have put their efforts on.

2.4.2 Approaches for ontology mapping

Different approaches to ontology mapping have been developed. Manual mappings between some large ontologies have been tried in recent years. In [RL02], researchers from OpenCyc integrated knowledge from resources like SENSUS, FIPS (Federal Information Processing Standards), pharmaceutical thesauri, WordNet, MeSH (Medical Subject Headings), CIA World Factbook, etc. into OpenCyc's knowledge base. Knowledge workers, such as those who are trained to develop OpenCyc ontologies, and domain experts, people who have expertise knowledge of a given domain, worked together for this task. To make the work more efficient, a dialogue based tool was developed to work with the domain experts, so that in some cases they could just input knowledge and answer questions prompted by the program, without having to know how to convert their input into OpenCyc's ontology language. In [NP03], mapping the noun synsets from WordNet to SUMO was carried out. Different relations between concepts are marked during the mapping process. For example, a noun from SUMO will be added at the end of the corresponding WordNet entry with a prefixed sign. Manual mapping like these is accurate and it can be saved for future use. The problem is that the size of ontologies can be very large and ontologies can keep growing, which requests a huge amount of continuous human efforts. Moreover, this approach is also restricted by the availability of domain experts' knowledge and its slow response time. More and more researchers are looking for ways to map ontologies semi-automatically or automatically.

String matching of concept names in two ontologies [Li04] is an effective alternative. Large amount of information can be processed very quickly and with some degree of accuracy. For example, in [Li04], the first step is to do a "whole term matching". The terms used in both ontologies are changed to lowercase and are used for string comparison. After the first step, a "word constituent matching" is performed for concepts that do not find a match yet. A term, which is a combination of several English words, for example, ArtificialIntelligence, ComputerScience, TeachingAssistant, is broken into separate words whenever a capital letter is encountered. Stop words like "the", "a", "in", etc. are filtered out. The remaining words are processed morphologically and are used as inputs to string matching. Terms such as "written-by" and "wrote", "meetingPlace" and to string matching. Terms such as “written-by” and “wrote”, “meetingPlace” and “PlaceOfMeeting” can be matched with one another in this way.

The next step is “Synset matching”, where WordNet [wn] is used. For each term in an ontology file that do not have a match yet, if its constituent words are in WordNet, then each of these words must belong to some synsets. Synset index numbers are recorded for each term. Two terms from two different ontologies are matched if they have the most synset index numbers in common. If by these three steps, a mapping still cannot be found for a term and this term has synsets in WordNet, corresponding SUMO mapping of its synset is searched. Using this SUMO mapping as a bridge, possible mapping to terms in the target ontology can be found. This requires the target term also has associated synsets, which has mappings in SUMO. Approaches like this usually involve complicated lexical analysis and a complete dictionary such as WordNet has to be consulted to make a correct mapping. Also, there are some situations that the string matching approach will not work at all, because one word can have different semantic meanings or word senses which are represented by different classes, possibly in different ontologies. For example, the word notebook can either be a kind of computer or be a kind of office supplies. A tool based on the string matching approach would map these two different classes as equivalent to one another.

Many researchers choose machine learning methods to solve the problem, especially text classification techniques [MG01, DMDDH02, SPF02], because the semantic meaning of a concept is contained in an exemplar that uses the concept in a desired context. Usually, such text exemplars for each concept or class in a given ontology (Onto$_A$) are manually prepared. Then a text classifier is trained using these data. To map a concept *C* defined in another ontology (Onto$_B$) to some concept in Onto$_A$, exemplars for *C* need to be collected and classified into the classifier of Onto$_A$, which itself was built from exemplars of its concept. Based on the initial classification results, algorithms such as [LG01] and [DMDDH02] can be used to carry out the further steps of ontology mapping. Text classification based ontology mapping is more efficient than manual mapping, and more powerful than string matching, because semantic meanings of apparently different strings can be analyzed by processing information contained in the provided exemplars. Here, the existence of exemplars for each concept and their relevancy to the concept they represent are the key factors to the effectiveness of this approach. However, finding sufficient quantity of high quality exemplars *manually* is time-consuming, and is thus the limiting factor of this approach.

2.5 Our proposal

The WWW is the richest information resource available anywhere in the world. There are tens of billions of web pages available on the WWW. There must be a sufficient number of documents on the WWW that explain a concept, describe the usage of a concept, or use the desired semantic meaning of a concept in some context. If there is a way to find such documents automatically and quickly, the limiting factor of the text classification based approach will be gone and we will be in a better position to solve the ontology mapping problem.

To achieve this goal, we can choose to use a search engine as part of the solution. A search engine is a killer application of the Internet. It indexes tens of billions of web pages accessible on the web and provides links of the most relevant ones to a query submitted by a user. Of course, a search engine would never know whether a document is relevant or not regarding a search query, only a human user can judge that. A search engine can only make its best guesses based on some very complicated algorithms. However, a good search engine would offer us the best combination of speed and search qualities verified by hundreds of millions human users, with the most relevant documents to a search query often appearing in the first few pages of its search results. Based on virtually unlimited text information offered by the Internet and the matured services provided by modern search engines, we propose the following hypothesis:

By using a concept defined in an ontology to form a search query and collecting web documents related to the concept from the WWW, we can get a good quantity of useable text exemplars.

To prove this hypothesis, we designed a tool to retrieve documents from the web with the help of a search engine and tried a number of different ways to process the documents downloaded before using them for text classification. We tested our tool by actually performing some preliminary ontology mapping experiments with small scale ontologies.

3. System Design

When describing the components of the system, we use Onto$_A$ to refer to the ontology in which we seek a mapping for a foreign concept and use Onto$_B$ to refer to the ontology which provides the semantic definition of the foreign concept. The system has the following main components:

1. A ***parser*** to parse ontology files in OWL format to form search queries.
2. A ***retriever*** to drive a web search engine with the queries generated by the parser and to retrieve a specified number of web pages based on the search results.

3. A ***processor*** to process the raw HTML documents obtained from the retriever to construct text files as exemplars for concepts in the ontologies.
4. A ***model builder*** to build a probabilistic model from the processed text files associated with Onto_A by calling a text classification software.
5. A ***calculator*** to feed the text files produced by the processor for concepts in Onto_B to the text classifier built for Onto_A, collects classification outputs and calculates initial mapping results as conditional probabilities.

We chose Google as our search engine and Rainbow [MC96] for our text classification. The structure of the system is shown below. Figure 3 shows the process from generating queries to obtaining text exemplars. Figure 4 shows the process from building a probabilistic feature model to calculating conditional probabilities for ontology mapping. Details of different components are explained in later sections.

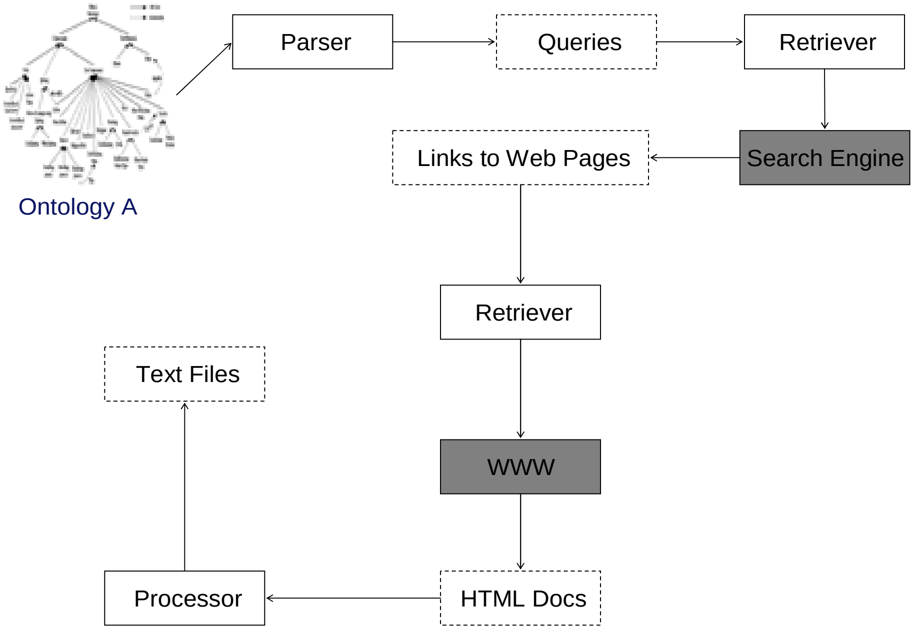


Figure 3 System components overview Part I

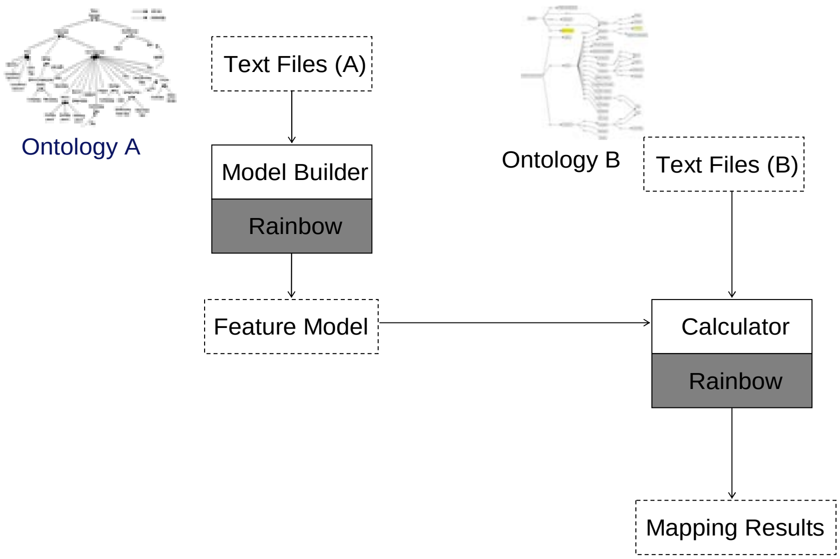
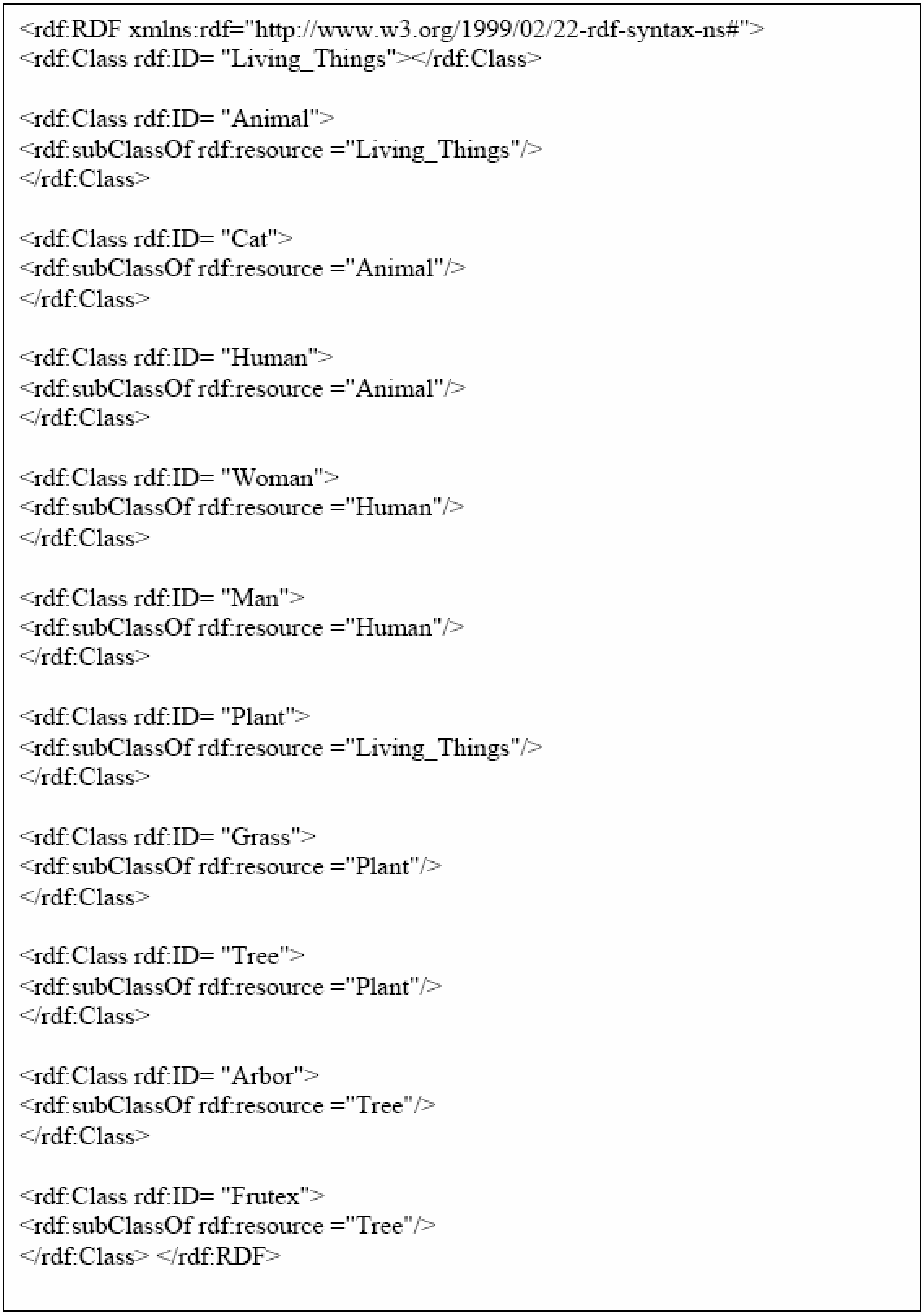


Figure 4 System components overview Part II

3.1. The parser

Given an ontology file, we parse the file to generate search queries for Google. To obtain better results, the query should contain more semantic information than just a class name. Because a word may have multiple senses or meanings, a query consisting of only the words of a concept's name may return web pages based on a more popular meaning of the word, which sometimes is not the particular meaning intended for the concept in the ontology. For example, in an ontology for food with a root class called "food", we may have a concept "apple", which is a subclass of "fruit". If we only use "apple" as the keyword, documents showing how to make an apple pie and documents showing how to use an iPod may both be returned with more documents for iPod due to its popularity on the web. Apparently, the documents using apple for its meaning in computer and entertainment fields are irrelevant to a subclass of fruit. To avoid this, when forming a query, we use **all the terms on the path from root class to the class in question** together as a query to send to the search engine. In the apple example, the query would be "food+fruit+apple" instead of "apple" itself alone. By doing so, the number of irrelevant documents returned can be greatly reduced. This kind of "word sense disambiguation" by adding additional semantically relevant terms into the search queries can be further extended to include the concept's properties, its subclasses, instances, and other semantically related items available in the ontology definition file. For our experiments we developed a simple LIVING_THINGS ontology for the biology domain. Figure 5 is the ontology in OWL format. The concept tree in Figure 6 shows relations between the concepts and Table 1 lists the queries generated for each class defined in the LIVING_THINGS ontology.



```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
<rdf:Class rdf:ID= "Living_Things"></rdf:Class>

<rdf:Class rdf:ID= "Animal">
<rdf:subClassOf rdf:resource ="Living_Things"/>
</rdf:Class>

<rdf:Class rdf:ID= "Cat">
<rdf:subClassOf rdf:resource ="Animal"/>
</rdf:Class>

<rdf:Class rdf:ID= "Human">
<rdf:subClassOf rdf:resource ="Animal"/>
</rdf:Class>

<rdf:Class rdf:ID= "Woman">
<rdf:subClassOf rdf:resource ="Human"/>
</rdf:Class>

<rdf:Class rdf:ID= "Man">
<rdf:subClassOf rdf:resource ="Human"/>
</rdf:Class>

<rdf:Class rdf:ID= "Plant">
<rdf:subClassOf rdf:resource ="Living_Things"/>
</rdf:Class>

<rdf:Class rdf:ID= "Grass">
<rdf:subClassOf rdf:resource ="Plant"/>
</rdf:Class>

<rdf:Class rdf:ID= "Tree">
<rdf:subClassOf rdf:resource ="Plant"/>
</rdf:Class>

<rdf:Class rdf:ID= "Arbor">
<rdf:subClassOf rdf:resource ="Tree"/>
</rdf:Class>

<rdf:Class rdf:ID= "Frutex">
<rdf:subClassOf rdf:resource ="Tree"/>
</rdf:Class> </rdf:RDF>
```

Figure 5 Living_Things ontology in OWL format

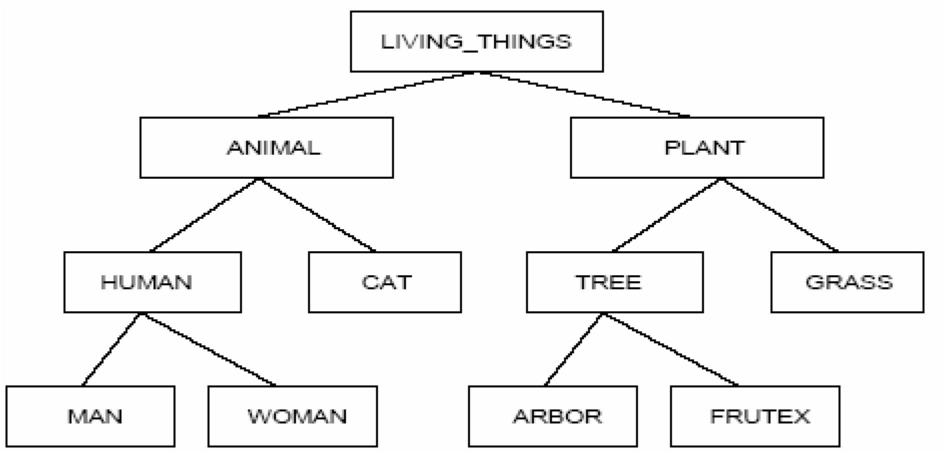


Figure 6 Structure of LIVING_THINGS ontology

Concepts	**Queries**
living+things	living+things
Animal	living+things+animal
Plant	living+things+plant
Cat	living+things+animal+cat
Human	living+things+animal+human
Man	living+things+animal+human+man
woman	living+things+animal+human+woman
Tree	living+things+plant+tree
Grass	living+things+plant+grass
Frutex	living+things+plant+tree+Frutex
Arbor	living+things+plant+tree+arbor

Table 1 A set of queries generated from LIVING_THIGNS ontology

It needs to be noted that, as shown in some of our experiments, queries that include too many terms of high specificity (e.g., some concepts in zoology or botany) may lead to very few or even empty search results.

3.2. The retriever

There are tens of billions documents available on the web. For a text classification problem, usually only a few dozens to a few hundreds exemplars for each concept are needed. For most of the experiments, we retrieved documents with the help of Google, because it is the easiest one to be integrated into our system and it is generally considered the best by human users, which makes us believe that its results are possibly the most relevant to the query among publicly available search engines. Although Google provided a programming API to download web pages listed in search results, we still decided to develop our own retriever program. This gives us the flexibility to experiment with search engines other than Google (for example, Clusty.com, a search engine which also clusters the search results [cl]) in some of our experiments.

The retriever takes a file containing queries generated by the parser, initiates a connection with a search engine, and sends a query in. Then it goes through the result pages one by one and extracts URLs of web pages listed as search results on each page. URLs of web pages listed as the search results can be identified by their special HTML coding patterns, so advertisements and other URLs on each page that are not search results can be escaped. After it collects a pre-specified number of URLs, it tries to download web pages at these URLs. In the current prototype implementation, only URLs starting with http:// and ending with .html, .htm or / are extracted because other file types, for example, doc or pdf will be difficult for our current processor to process. Sometimes, the retriever cannot download a web page due to connection timeout or the web page being deleted but still cached by the search engine. All the HTML files obtained through a query for a particular class are saved in a directory using the query as the directory name and will be used by the processor to generate exemplars for that class.

3.3. The processor

Documents collected by the retriever are HTML files. They contain HTML tags, images, texts and script programs etc. These raw data have to be processed before being fed into a text classifier. The processor will remove all HTML tags, image files, script programs, etc. Also removed are hyperlinks, which may contain some useful semantic information. For example, for the query “living+things+animal”, the document ranked highest by Google is at http://www.fi.edu/tfi/units/life/classify/classify.html. Many animal names are represented using hyperlinks in this page. But more often a webpage will contain a certain amount of links to other irrelevant pages and websites, for example, online advertisements, other services and information provided by the hosting website which are not related to the topic of the current page, etc. Since the retriever can easily retrieve a huge amount of relevant documents from the web, we can afford to be more selective in the process and do not have to worry about losing a very small amount of potentially useful information when removing hyperlinks.

After the above steps, what we have are a large number of pure text files for each concept, contained in one directory per concept. The processor now has a optional step for these files: keeping only the *sentences* where a word in the query appears and delete all other sentences. Since not every part of the text file is necessarily relevant to the class in question, this step may help remove irrelevant information and keep only the most closely relevant texts. Text files processed with and without this option are both used in our experiments and the results are compared. The processor can also choose to keep *paragraphs*, rather than sentences in which query words appear.

3.4. Bayes rule and the naïve Bayes text classifier

There are quite a few text classification algorithms available for ontology mapping, for example, naïve bayes, TF-IDF, and support vector machines, etc. We choose to use naïve bayes text classification in our research because it is a simple algorithm and it shows good performance in many evaluations [Mi97]. We also thought that it would give an accurate value for the conditional probability for each class used to train the classifier, given text exemplars of a new class.

3.4.2 Bayes rule

A naïve Bayes classifier is based on Bayes rule. Suppose we have event A and B in our event space E, Bayes rule tells us that

$$P(A \mid B) * P(B) = P(B \mid A) * P(A) = P(A, B)$$

Where

- $P(A)$ is the *prior* or *marginal* probability. It describes our belief that event A will happen.
- $P(B)$ is the prior probability of even B.
- $P(A|B)$ is the *conditional* probability of A, given B. It describes our belief that if event B happens, event A will happen.
- $P(\mathrm{B}|A)$ is the conditional probability of B, given A.
- P(A, B) is the *joint* probability of event A and event B. It describes our belief that both event A and event B happen at the same time.

Bayes rule explains the relations between prior probability, conditional probability and joint probability. It is a powerful tool for probability reasoning. Especially when $P(A \mid B)$ is useful, but unknown and hard to obtain directly, we can use

$$P(A \mid B) = P(B \mid A) * P(A) / P(B)$$

to calculate $P(A \mid B)$, the posterior probability of A, given B if other variables in Bayes rule are available, which is precisely the situation of a text classification problem.

3.4.2 Naïve Bayes text classifier

In a text classification problem, we need to decide among a set of mutually exclusive categories or events C_1, C_2, … C_n, to which category a new document d should belong. This can be determined by which category has the greatest posterior probability, given d, i.e., $\max_i(P(C_i \mid d)| \ i = 1, \ldots, n)$.

$P(C_i \mid d)$ is hard to calculate directly, so we apply Bayes' rule here,

$$P(C_i \mid d) = P(d \mid C_i) * P(C_i) / P(d)$$

Since $P(d)$, the prior probability of d, also called a normalizing constant, is the same for every C_i, the classification is thus determined by

$$\max_i(P(d \mid C_i) * P(C_i)), \qquad (1)$$

because what we are interested here is not the absolute values of the posteriors but their ranking among all categories. Here $P(C_i)$, the prior probability of category i, can be estimated by the ratio of the number of exemplars for category C_i and the number of exemplars of all categories (if these exemplars are randomly drawn samples) or some other methods [Mi97].

Probability $P(d \mid C_i)$ is more difficult to estimate. Naïve Bayes classification does it based on word frequencies in each category for a predetermined set of words. Let d contain *m* distinct words $d = \{w_1, \ldots, w_m\}$, assuming that whether a word appears in a category is independent of other words in that category, then we have

$$P(d \mid C_i) = P(w_1, \ldots, w_m \mid C_i) = \Pi_{j=1}^{m} P(w_j \mid C_i) \tag{2}$$

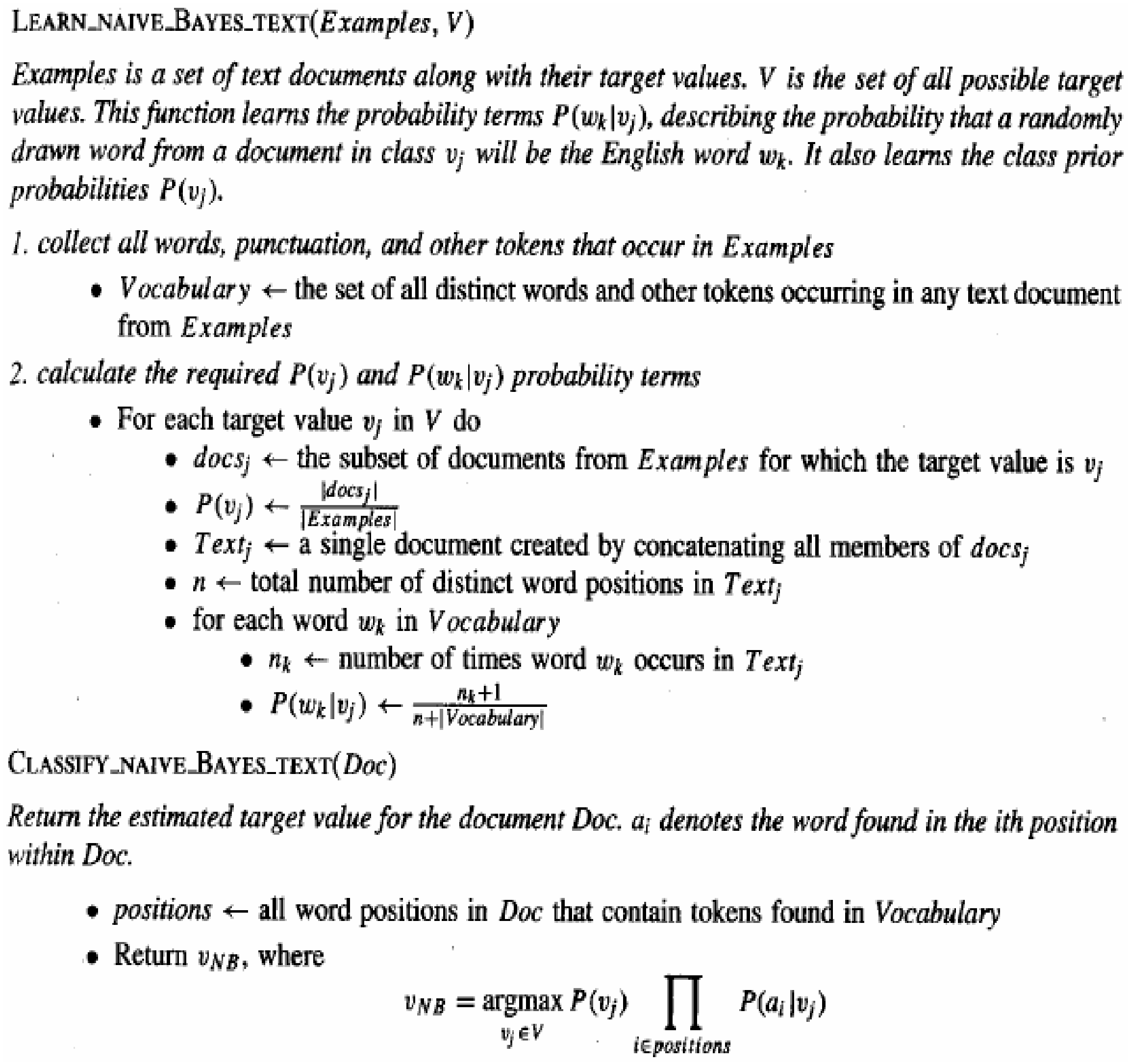
Probabilities $P(w_j \mid C_i)$ for all categories C_i and all words w_j form the probabilistic feature model for these categories, and they can be easily learned from exemplars associated with each C_i. Then the classifier, which combines the decision rule of (1) and the model (2), is given as

$$\max_i (P(C_i) \Pi_{j=1}^{m} P(w_j \mid C_i)) \text{ .} \tag{3}$$

If needed, the actual posterior probability $P(C_i \mid d)$ can then be computed by normalizing $P(C_i , d) = P(d \mid C_i) * P(C_i)$ where $P(d \mid C_i)$ is given by (2):

$$P(C_i \mid d) = \frac{P(d \mid C_i) P(C_i)}{\sum_j P(d \mid C_j) P(C_j)} \tag{4}$$

Note that the independence assumption in naïve Bayes classifier does not hold in general. Despite of this, good performance is still achieved. Figure 7 taken from [Mi97] shows the general algorithm of a naïve Bayes text classifier. Details of naïve Bayes classifiers can be found in [Mi97]



```
LEARN_NAIVE_BAYES_TEXT($Examples, V$)

Examples is a set of text documents along with their target values. $V$ is the set of all possible target values. This function learns the probability terms $P(w_k|v_j)$, describing the probability that a randomly drawn word from a document in class $v_j$ will be the English word $w_k$. It also learns the class prior probabilities $P(v_j)$.

1. collect all words, punctuation, and other tokens that occur in $Examples$
    • $Vocabulary$ ← the set of all distinct words and other tokens occurring in any text document from $Examples$
2. calculate the required $P(v_j)$ and $P(w_k|v_j)$ probability terms
    • For each target value $v_j$ in $V$ do
        • $docs_j$ ← the subset of documents from $Examples$ for which the target value is $v_j$
        • $P(v_j) \leftarrow \frac{|docs_j|}{|Examples|}$
        • $Text_j$ ← a single document created by concatenating all members of $docs_j$
        • $n$ ← total number of distinct word positions in $Text_j$
        • for each word $w_k$ in $Vocabulary$
            • $n_k$ ← number of times word $w_k$ occurs in $Text_j$
            • $P(w_k|v_j) \leftarrow \frac{n_k+1}{n+|Vocabulary|}$

CLASSIFY_NAIVE_BAYES_TEXT($Doc$)

Return the estimated target value for the document Doc. $a_i$ denotes the word found in the ith position within Doc.

    • $positions$ ← all word positions in $Doc$ that contain tokens found in $Vocabulary$
    • Return $v_{NB}$, where
          $v_{NB} = \underset{v_j \in V}{\operatorname{argmax}} P(v_j) \prod_{i \in positions} P(a_i|v_j)$
```

Figure 7 General Algorithm for a naïve Bayes classifier

We chose Rainbow as our text classifier. It is robust, fast and it implements naïve Bayes text classification algorithm. However, because of the conditional independence assumption, Rainbow and other naïve Bayes text classifiers tend to produce extreme values (0 and 1) for the conditional probabilities according to (4). The classifier will assign 1 or a value very close to 1 to a category that it a clear winner for the new document and assign 0 or a value very close to 0 to other categories. This is certainly good enough if one only wants to get a right classification result. However, our purpose is to use the classifier to obtain the conditional probability of a concept in Onto$_A$, given a new concept in Onto$_B$, and hope to use this value as a basis to measure the semantic similarity between these two concepts. In other words, what we want are more accurate posteriors, not those that are distorted by the independence assumption of naïve Bayes. For this purpose we developed a calculator, given in Subsection 3.6 shortly, which uses Rainbow to calculate this conditional probability (which can be any value between 0 and 1).

3.5. The model builder

A naïve Bayes classifier requires the predefined categories C_1, C_2, …, C_n to be exhaustive regarding to the domain and mutually exclusive to one another, so that the probability results can be correctly sum to 1. In our system, classification categories are closely related to ontology concept classes. Our model builder allows one to select concept classes in different ways when forming these classification categories.

For simplicity, this work only considers OWL ontology files that can be viewed as a concept tree based on the subClassOf property. The leaf classes in such a tree are assumed to be mutually exclusive to one another and exhaustive regarding to the root class. By leaf classes, we mean those classes that do not have a subclass.

The default behavior of the model builder is to use all leaf classes in an ontology as the classification categories, and each of which is associated with a set of text exemplars generated from the processor module. Then the model builder calls Rainbow to build a probabilistic feature model for these categories. This model will then be used by the calculator to calculate the conditional probability of each leaf class, given exemplars of a foreign class.

Besides the default behavior, the model builder has an option to build a model for each class in Onto$_A$ except the root. Two exhaustive and mutually exclusive categories A^+ and A^- are created by the model builder for class A in Onto$_A$. A^+ is associated with exemplars for that class, and A^- is associated with exemplars for the complement of that class, which are taken from classes that are not *A*, not *A*'s ancestors nor *A*'s successors in the ontology tree. The model builder then uses these two categories to build a model. This option is not applicable to the root class, because the root does not have a complement in the context. For example, consider the class "CAT" in the LIVING_THINGS ontology tree shown again in Figure 8. The exemplars for its complement "not CAT" would include exemplars found for all classes except "CAT" and except its ancestors "ANIMAL" and "LIVING_THINGS", which are classes in shadow. The model builder can also restrict the scope of complement in order to control the number of exemplars when there are too many such shadowed classes. For example, we can restrict the complement to be only with respect to the parent of the class in question. Then the category of "not-CAT" can only has exemplars found for "HUMAN" and its descendents, shown in Figure 8 as the shadowed classes in a circle. Alternatively, exemplars for these two categories can be retrieved directly using two queires (e.g., "living+things+animal+cat" and "living+things+animal+-cat" in the "CAT" example).

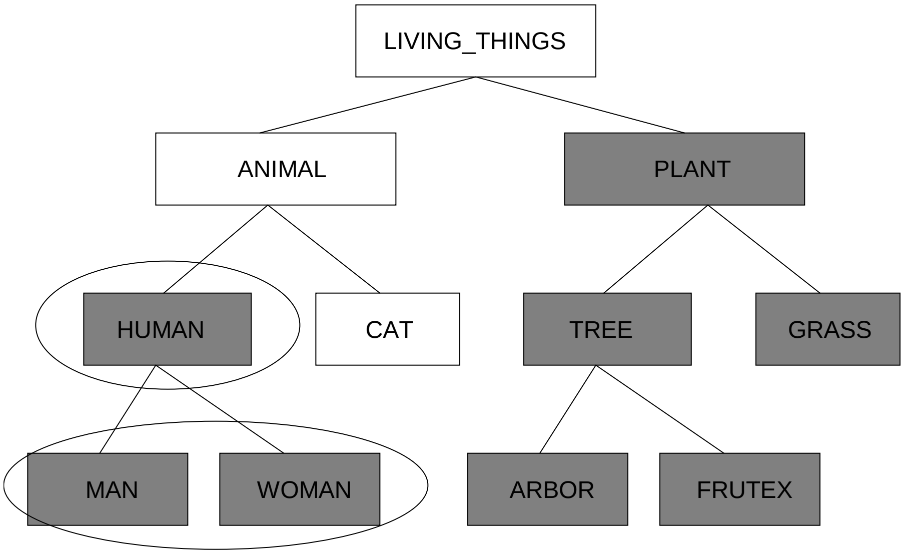


Figure 8 Using exemplars from complement classes to build model

3.6. The calculator

The tasks for the calculator are to (1) feed exemplars of a concept *C* of Onto$_B$ one by one to Rainbow, which performs classification using the model of Onto$_A$, which has already been built by Rainbow, (2) keep record of the classification results for each exemplar, (3) calculate average results grouped by categories in the model as the conditional probabilities, and (4) write a summary report. It can also perform some additional calculations like estimating mapping results for non-leaf classes.

One way to obtain conditional probabilities is to concatenate all the exemplars of a foreign concept into one piece and feed this to Rainbow. As we have discussed, because of the naïve conditional independence assumption, Rainbow and text classifiers alike tend to output extreme values such as 0 and 1. Such values are not true conditional probabilities that would be useful in advanced mapping algorithms such as the one in research of [DPP05] and [DPPY05]. To avoid such extreme values, we treat each exemplar as a sample of a foreign concept and feed the exemplars one by one to Rainbow and average the results to obtain conditional probabilities. Here is an example to illustrate the way how an average classification result as a conditional probability for a class is calculated. For example, APC (Armored Personnel Carrier) is a class in WeaponsB.n3, an ontology file describing some knowledge in the weapons domain. For the simplicity of our example, suppose WeaponsA.n3, another ontology file for the same domain, has three classes which do not have subclasses. They are TANK-VEHICLE, AIR-DEFENSE-GUN, and SAUDI-NAVAL-MISSILE-CRAFT. We build a model using these three classes as classification categories. To calculate the conditional probabilities given class APC, we classify each exemplar of APC against the model. Table 2 below gives the classification results with 200 exemplars of APC.

Categories in WeaponsA.n3	*Num. of exemplars falling in each category*
TANK-VEHICLE	170
AIR-DEFENSE-GUN	20
SAUDI-NAVAL-MISSILE-CRAFT	10

Table 2 Example classification results

Then by taking the average, the conditional probability P(TANK-VEHICLE | APC) = 170 /200= 0.85, and similarly, 0.1 and 0.05 for the other two classes. Table 3 shows an actual report that the calculator generated for one of the experiments with the WEAPONS ontology when the classifier is trained with 63 leaf classes from WeaponsA.n3 and the foreign class is APC coming from WeaponsB.n3. Each class contains averagely 100 exemplars.

APC	
SELF-PROPELLED-ARTILLERY	0.357180681
TANK-VEHICLE	0.277139274
ICBM	0.10423636
MRBM	0.080615147
TOWED-ARTILLERY	0.054724102
SUPPORT-VESSEL	0.023265054
PATROL-CRAFT	0.019570325
MOLOTOV-COCKTAIL	0.015032411
TORPEDO-CRAFT	0.013677696
SUPER-ETENDARD	0.009856519
MORTAR	0.00772997
AIR-DEFENSE-GUN	0.002997109
PATROL-COMBATANT	0.002846281
TORPEDO	0.002687264
TORNADO	0.002641316
HY-4-C-201-MISSILE	0.001898627
ANTI-RADAR-MISSILE	0.001868698
AIR-TO-AIR-MISSILE	0.00175536
MINE-WARFARE-VESSEL	0.001714463
TORPEDO-CRAFT	0.001589625
SS-N-22-SUNBURN-LCM	0.001560581
SILKWORM-MISSILE	0.001317753
SURFACE-TO-AIR-MISSILE	0.001144551
AIR-TO-SURFACE-MISSILE	0.000887746
NODONG-2-MISSILE	0.000882243
M-9-MISSILE	0.000863944
CSS2-MISSILE	0.000830065
AIRCRAFT-CARRIER	0.000721183
AMPHIBIOUS-VESSEL	0.000605727
MINE-WARFARE-VESSEL	0.000579844
CORVETTE	0.000521464
SMALL-ARMS	0.000520033
CHEMICAL-WEAPON	0.000518254
NUCLEAR-WEAPON	0.000495419
BIOLOGICAL-WEAPON	0.000493164

CRUISER	0.000456558
AL-HUSSEIN-MISSILE	0.000453844
FRIGATE	0.000419076
SAUDI-NAVAL-MISSILE-CRAFT	0.000413961
AL-FATTAH-MISSILE	0.000384428
SCARAB-MISSILE	0.000347386
NODONG-1-MISSILE	0.000335111
ARTILLERY-SHELL	0.000309594
SILKWORM-MISSILE-MOD	0.000274119
AS-11-KILTER-ALCM	0.000268002
PRINCIPAL-SURFACE-COMBATANT	0.00021418
MACHINE-GUN	0.000211772
MOLOTOV-COCKTAIL	0.000187578
TRUCK-BOMB	0.000171675
AS-9-KYLE-ALCM	0.000156403
ARABIL-100-MISSILE	0.000111953
AL-HIJARAH-MISSILE	7.65E-05
OGHAB-MISSILE	7.12E-05
BADAR-2000	4.28E-05
YJ-2-C-802-LCM	3.86E-05
SCUDB-MISSILE	2.21E-05
SAQR-2000	2.17E-05
CSS8-MISSILE	1.69E-05
RGM-84A-HARPOON-SLCM	1.37E-05
SCUDC-MISSILE	1.11E-05
MUSHAK120-MISSILE	5.10E-07
TAMMOUZ1-MISSILE	5.10E-07
ZELZAL2-MISSILE	5.10E-07

Table 3 Conditional probability given APC calculated by the calculator

4. Experiments and results

A several ontologies of different sizes have been used in many of our experiments. To be concise, we only report here some representative experiments which involved two sets of ontologies. The first set of experiments uses a small ontology whose structure is shown in Figure 6. We performed text classification between classes within this ontology and also with some foreign concepts.

The second set of experiments uses two ontologies WeaponsA.n3 and WeaponsB.n3 taken from I^{3}Con2004 [i3c]. Each of these two ontologies contains over 80 classes, not large to be included in this thesis. Their complete descriptions can be found in [i3c].

The system was implemented on a Linux system. The retriever is coded in Java and the other components are done by Perl. Different components are glued together by shell scripts. The entire process from parsing, generating queries, to collecting exemplars, building models and calculating results is fully automated. The runtime of this process depends on how many text files are to be processed. Usually the retriever takes most of the time. It could take nearly 3 minutes to download 50 documents. However, this can be improved by using advanced programming techniques such as multithreading and setting the timeout for connections shorter. Processing documents, building model and calculating conditional probabilities is comparatively a lot faster. For a process involving 100 documents, it usually only takes a few seconds for the processor, the model builder and the calculator to finish their jobs. Some utility tools, written in Perl, have also been developed for monitoring the process and post-processing the results, including sorting, formatting for EXCEL, and updating the conditional probability of non-leaf classes in Onto$_A$ based on the conditional probabilities of its leaf classes given a foreign concept, etc.

4.1. Results for weapons ontologies

When generating a query for a class in the default mode, the parser will use all the classes from root class to the class in question. For weapons ontologies, because of their high specificity, we decided to let the parser generate shorter queries, using only the names of the class itself and its parent class. This allows the search engine to return more results instead of none or very few for some of the full-path queries.

Onto$_A$, WeaponsA.n3 has more than 60 leaf classes. The model builder would run in default mode, which would build a model using these leaf classes as classification categories. The retriever collected on average 100 exemplars for each class. The processor was executed in two different ways for comparison as discussed earlier: one is to use the default mode to obtain pure text exemplars; the other is to only keep sentences in which any of the search keywords appear as exemplars.

There are 9 classes in WeaponsB.n3 that do not appear in WeaponsA.n3. We try to find a mapping for each of them in WeaponsA.n3. The relations between these 9 classes are shown in Figure 9 and Figure 10. The shadowed blocks are these classes that need to be mapped.

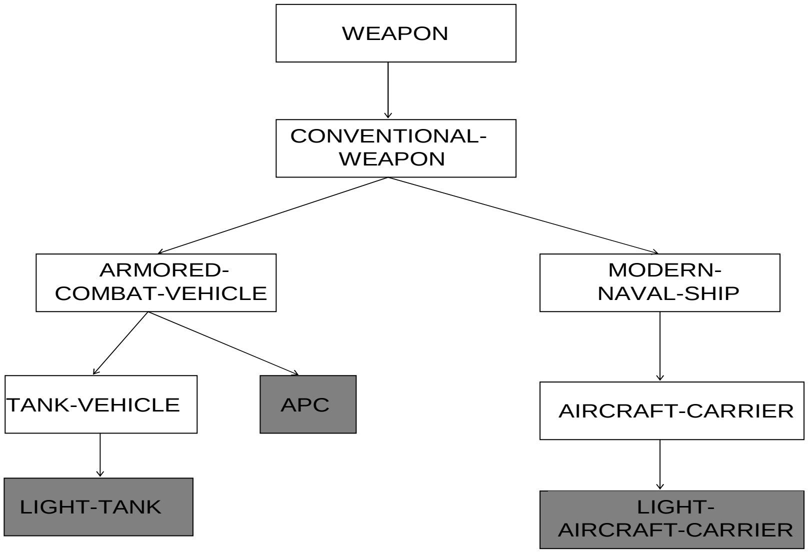


Figure 9 Classes in WeaponsB.n3 that are not in WeaponsA.n3 (I)

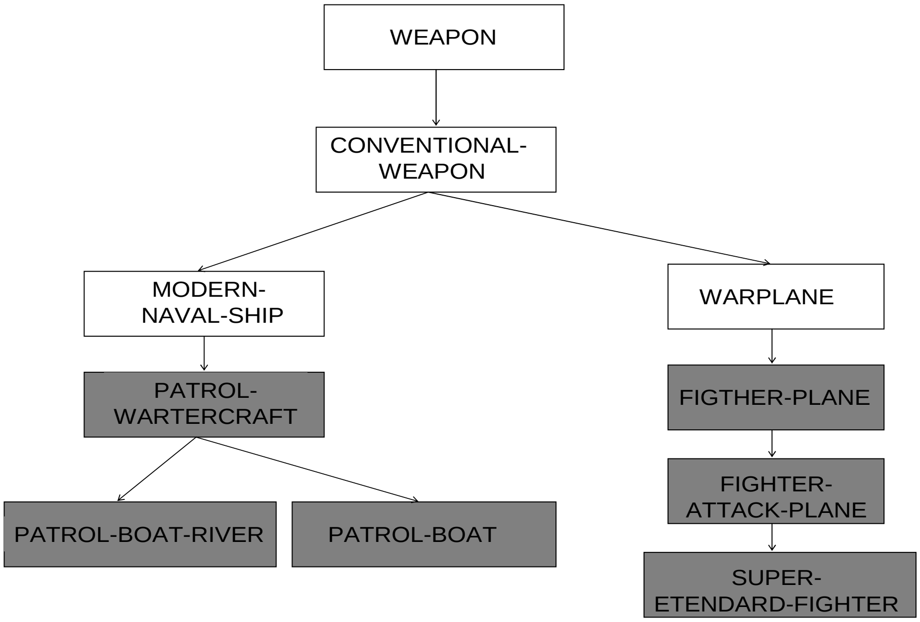


Figure 10 Classes in WeaponsB.n3 that are not in WeaponsA.n3 (II)

These 9 classes and their manually selected desired mapping leaf classes in WeaponsA.n3 are listed in Table 4.

Classes from WeaponsB.n3	***Desired leaf class mappings***
LIGHT-AIRCRAFT-CARRIER	AIRCRAFT-CARRIER
APC	TANK-VEHICLE
SUPER-ETENDARD-FIGHTER	SUPER-ETENDARD
FIGHTER-ATTACK-PLANE	SUPER-ETENDARD
PATROL-WATERCRAFT	PATROL-CRAFT
PATROL-BOAT-RIVER	PATROL-CRAFT
PATROL-BOAT	PATROL-CRAFT
LIGHT-TANK	TANK-VEHICLE
FIGHTER-PLANE	SUPER-ETENDARD

Table 4 Classes and their desired mappings

The conditional probabilities obtained are given in Table 5. Totally 9 times of mapping were performed for the 9 classes. For space limitation, here for each time of the mapping, we only list the class that has the highest probability obtained instead of the complete results for over 60 leaf classes. The first column contains classes from WeaponsB.n3, which we are seeking a mapping for. The second and the third columns are the classes in WeaponsA.n3 with the highest conditional probability obtained by using a whole file as an exemplar. The last two columns are results obtained by using only sentences containing keywords as an exemplar. MRBM in column 4 stands for "Medium-Range Ballistic Missiles".

New Classes	***Whole file***	***Prob***	***Sentences with Keywords***	***Prob***
LIGHT-AIRCRAFT-CARRIER	AIRCRAFT-CARRIER	0.65	AIRCRAFT-CARRIER	0.57
APC	SILKWORM-MISSILE-MOD	0.46	SELF-PROPELLED-ARTILLERY	0.36
SUPER-ETENDARD-FIGHTER	SILKWORM-MISSILE-MOD	0.66	(BALLISTIC-MISSILE) MRBM	0.51
FIGHTER-ATTACK-PLANE	SILKWORM-MISSILE-MOD	0.83	(BALLISTIC-MISSILE) MRBM	0.38
PATROL-WATERCRAFT	SILKWORM-MISSILE-MOD	0.28	PATROL-CRAFT	0.52
PATROL-BOAT-RIVER	SILKWORM-MISSILE-MOD	0.65	PATROL-CRAFT	0.54
PATROL-BOAT	SILKWORM-	0.51	PATROL-	0.66

	MISSILE-MOD		CRAFT	
LIGHT-TANK	SILKWORM-MISSILE-MOD	0.56	TANK-VEHICLE	0.3
FIGHTER-PLANE	AIRCRAFT-CARRIER	0.49	MRBM	0.38

Table 5 Results comparison by showing classes with highest conditional probability

If we simply judge the accuracy of the mapping by looking at the class that has the highest conditional probability, it is easy to see that when using a whole processed web document as an exemplar, only LIGHT-AIRCRAFT-CARRIER is correctly mapped. The accuracy is 11%. However, when using only sentences containing keywords, the results are improved significantly. The accuracy is 56% in this case. There are four classes, APC, FIGHTER-PLANE, FIGHTER-ATTACK-PLANE, and SUPER-ETENDARD-FIHTER, whose desired mapping classes does not have the highest conditional probability.

However, for class APC, its desired mapping class TANK-VEHICLE has the second highest conditional probability, which is 0.28, very close to the highest one, SELF-PROPELLED-ARTILLERY, which is also very related to TANK and APC. The fact that SELF-PROPELLED-ARTILLERY is matched with APC on one hand shows us that the exemplars collected and processed by the system preserve the semantic meaning of a concept quite well. On the other hand it also shows us that though text classification method and conditional probability can tell how related two concepts are, they cannot necessarily tell if they are equivalent. Because that two concepts are closely related does not mean that they are semantically identical or similar. This is a hard problem for our future research.

For class SUPER-ETENDARD-FIGHTER, its desired mapping class SUPER-ETENDARD also has the second highest conditional probability, which is 0.21. For the other two FIGHTER classes, the results are not good. We think one reason is SUPER-ETENDARD is the only leaf node in WeaponsA.n3 that represents a plane (violation of exhaustive assumption for categories). It is possible that it is indeed not a good mapping for some of the plane classes from WeaponsB.n3. For testing, we added a class WAR-PLANE-OTHER under the class WARPLANE in WeaponsA.n3, containing exemplars retrieved with a search query “WARPLANE+-SUPER+-ETENDARD” and performed the classification process again. This time class FIGHTER-PLANE is mapped to its desired class WARPLANE-OTHER with the highest conditional probability of 0.41. This shows that adding a complement class sometimes helps when the given ontology is not a complete model of the domain knowledge. Moreover, in this case, FIGHTER-PLANE is a super class of the other two classes (FIGHTER-ATTACK-PLANE and SUPER-ETENDARD-FIGHTER). The fact that a super class can be correctly mapped will make the mapping of its sub classes easier.

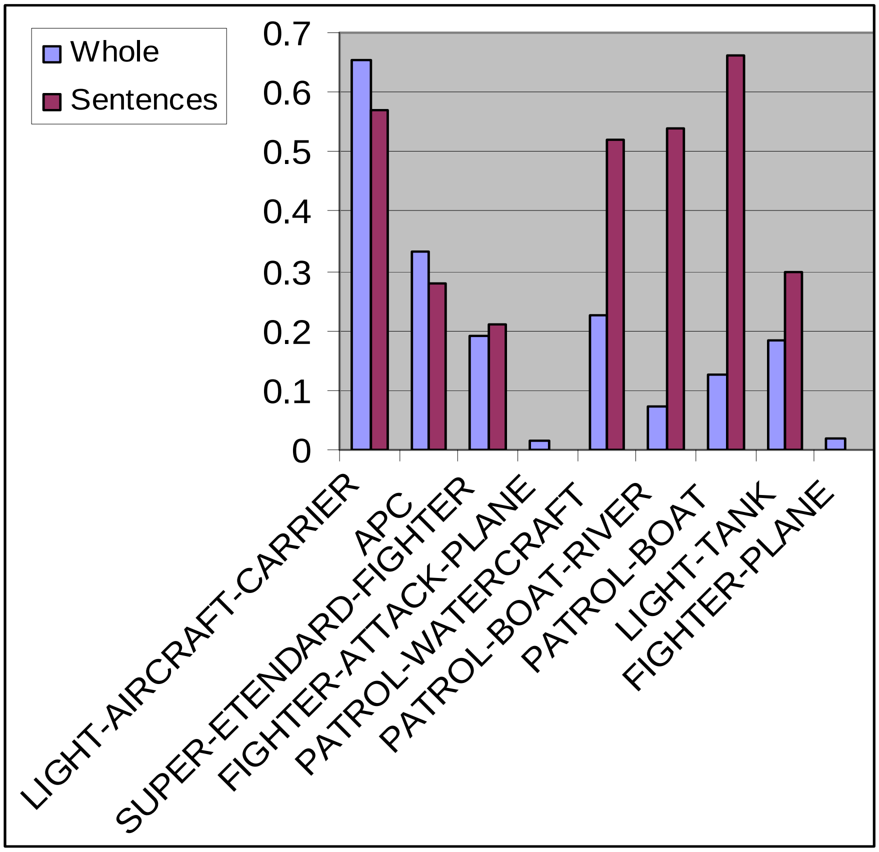


Figure 11 Comparison of different processing methods

Figure 11 shows a further comparison of the two methods for the processor module. The x-axis lists the classes from WeaponsB.n3 (the 1st column in Table 4). The y-axis represents the conditional probability of a desired mapping class in WeaponsA.n3 (the 2nd column in Table 4). Except class APC, FIGHTER-PLANE and FIGHTER-ATTACK-PLANE, all desired mappings are correctly identified by only keeping sentences containing keywords in an exemplar. This processing method filters out noisy information, which results in a better classification.

In the search results returned by a search engine, the most relevant documents to a query are always listed first. So the first 50 documents listed in the search results should be more relevant to the queries than the first 100. To verify this, we changed the number of exemplars the system use for each class and did some comparison experiments. Table 6 is a comparison of results obtained from the system by using the first 50 exemplars and the first 100 exemplars for each concept. These exemplars are obtained by processing search results downloaded in that order with the processor in default mode. Table 7 shows the same comparison but the exemplars are processed at a sentence level.

New Classes	***Group-whole-50***	***Prob***	***Group-whole-100***	***Prob***
LIGHT-AIRCRAFT-CARRIER	SILKWORM-MISSILE-MOD	0.60	AIRCRAFT-CARRIER	0.65
APC	SILKWORM-MISSILE-MOD	0.65	SILKWORM-MISSILE-MOD	0.46
SUPER-ETENDARD-FIGHTER	SILKWORM-MISSILE-MOD	0.74	SILKWORM-MISSILE-MOD	0.66
FIGHTER-ATTACK-PLANE	SILKWORM-MISSILE-MOD	0.83	SILKWORM-MISSILE-MOD	0.83
PATROL-WATERCRAFT	SILKWORM-MISSILE-MOD	0.64	SILKWORM-MISSILE-MOD	0.28
PATROL-BOAT-RIVER	SILKWORM-MISSILE-MOD	0.89	SILKWORM-MISSILE-MOD	0.65
PATROL-BOAT	SILKWORM-MISSILE-MOD	0.64	SILKWORM-MISSILE-MOD	0.51
LIGHT-TANK	SILKWORM-MISSILE-MOD	0.62	SILKWORM-MISSILE-MOD	0.56
FIGHTER-PLANE	SILKWORM-MISSILE-MOD	0.80	AIRCRAFT-CARRIER	0.49

Table 6 Comparison between different numbers of exemplars (whole)

New Classes	*Group-sentence-50*	*Prob*	*Group-sentence-100*	*Prob*
LIGHT-AIRCRAFT-CARRIER	AIRCRAFT-CARRIER	0.44	AIRCRAFT-CARRIER	0.57
APC	TANK-VEHICLE	0.54	SELF-PROPELLED-ARTILLERY	0.36
SUPER-ETENDARD-FIGHTER	HY-4-C-201-MISSILE	0.4	MRBM	0.51
FIGHTER-ATTACK-PLANE	ICBM	0.19	MRBM	0.38
PATROL-WATERCRAFT	PATROL-CRAFT	0.49	PATROL-CRAFT	0.52
PATROL-BOAT-RIVER	PATROL-CRAFT	0.36	PATROL-CRAFT	0.54
PATROL-BOAT	PATROL-CRAFT	0.37	PATROL-CRAFT	0.66
LIGHT-TANK	TANK-VEHICLE	0.59	TANK-VEHICLE	0.3
FIGHTER-PLANE	MRBM	0.38	MRBM	0.38

Table 7 Comparison between different numbers of exemplars (keyword sentence)

Here we use Group-whole-50 to refer to the group of experiments using a whole document as an exemplar and only using the first 50 for each concept, and use Group-sentence-50 to refer to the group of experiments using a document processed at a sentence level as an exemplar and only using the first 50 for each concept. Group-whole-100 and Group-sentence-100 have similar meanings but different in number. The accuracy comparison of the mappings of these four groups is shown in table 8.

Groups of experiments	***Mapping accuracy judged by desired class mapped***
Group-whole-50	0%
Group-whole-100	11%
Group-sentence-50	67%
Group-sentence-100	56%

Table 8 Comparison of mapping accuracy of different groups of experiments

From the results, we can see that experiments using the first 50 exemplars for each class have better results when the exemplars are processed at a sentence level. Class APC is correctly mapped in Group-sentence-50 but not in Group-sentence-100. However, the conditional probabilities obtained in Group-sentence-50 are averagely a lot lower than those obtained in Group-sentence-100, which may have a negative effect on the further calculations based on these conditional probabilities. So using less exemplars may not always be a good choice.

4.2. Results for LIVING_THINGS ontology

To gain further insights of this approach, we conducted the following additional experiments using the LIVING_THINGS ontology shown in Figure 5.

(1) Obtain P(MAN | HUMAN) and P(WOMAN | HUMAN). We expect both posteriors to be around 0.5

(2) Given a new, foreign concept GIRL, build a model with classes ANIMAL and PLANT as the set of mutually exclusive and exhaustive classification categories, and perform classification. If class GIRL is mapped to class ANIMAL, then build a model with Class HUMAN and CAT as the categories, and using GIRL exemplars classified into ANIMAL class to perform the classification. Finally repeat the process at the third level with class MAN and WOMAN to see how well GIRL can be mapped to WOMAN. This process is shown in Figure 12.

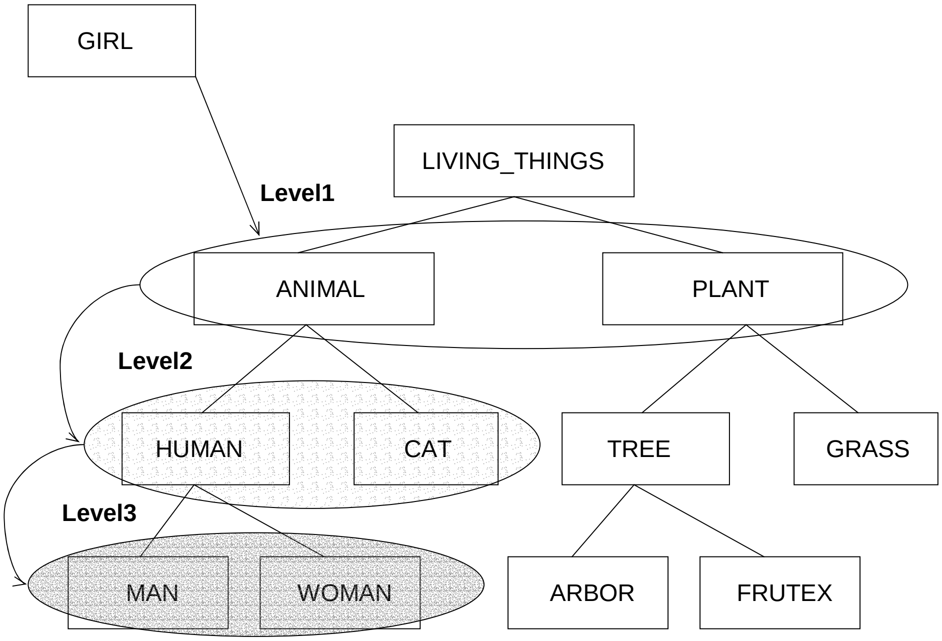


Figure 12 Experiment (2) with LIVING_THINGS ontology

The system performed these experiments automatically. Extensive experiments were done with varying parameters of the system. For example, changing the number of documents to be downloaded, adding different class properties to generate more informed queries, and using different modes of the processor module, etc. Table 9 shows results comparison for experiment (1) with different numbers of exemplars used. The exemplars are processed at a sentence level.

Conditional Probability	***Using first 50 exemplars***	***Using first 100 exemplars***	***Using first 200 exem-plars***
P(MAN | HUMAN)	0.75	0.58	0.62
P(WOMAN | HUMAN)	0.24	0.41	0.38

Table 9 Results of experiment (1)

Our expectation is that both P(MAN | HUMAN) and P(WOMAN | HUMAN) are to be 0.5. The results show that using 100 exemplars performs the best among the three groups. Also considering the experiments with WEAPONS ontology, we can see using fewer exemplars (these are also ranked the highest) sometimes achieves good results, sometimes not. We should not always depend on this single parameter. Table 10 shows results for experiment (2) using 200 exemplars processed at a sentence level.

P(ANIMAL | GIRL)	0.76
P(PLANT | GIRL)	0.23
P(HUMAN | GIRL)	0.70
P(CAT | GIRL)	0.30
P(MAN | GIRL)	0
P(WOMAN | GIRL)	1

Table 10 Results of Experiment (2) with 200 exemplars

What is disturbing is that Class CAT has a comparatively high conditional probability given GIRL (P (CAT | GIRL) = 0.3). One reason for this anomaly is that words like human, man, woman and girl often appear in web pages associated with class CAT because cats have such close relations with human beings (sometimes cat is even used to describe a human). Manually inspecting the exemplars confirms this reason.

The "cat" problem shows that even though the parser generates an informed query and the processor is able to further process exemplars at a sentence level, the exemplars may still be far from perfect. This problem was further confirmed by an additional experiment in which DOG (another domesticated animal) and PYCNOGONID (a kind of sea spider) were added into the ontology as subclasses of ANIMAL. Most of the exemplars of GIRL went to Dog, and none to PYCNOGONID as shown in Table 11.

P(DOG | GIRL)	0.56
P(CAT | GIRL)	0.01
P(HUMAN | GIRL)	0.43
P(PYCNOGONID | GIRL)	0

Table 11 Results with additional classes (200 exemplars each class)

We conjecture that, although all exemplars for CAT or DOG taken as a whole are closely related to GIRL, it is different at the level of individual exemplars, some are close but others are not. The "cat and dog" problem can then be solved if we can separate exemplars that truly reflect the intended semantics of CAT or DOG from those that are not. As a first step, we have tried to perform clustering on exemplars of each class in the hope that one of the clusters would contain those truly relevant exemplars.

We have tried to replace Google as the retriever's search engine by a clustering search engine Clusty.com which automatically clusters search results based on some text clustering algorithm. Then the largest cluster for each class returned by Clusty.com is used as exemplars. Even though the exemplars are not processed at a sentence level, results are a lot better regarding to P(CAT | GIRL) as shown in Table 12.

P(ANIMAL | GIRL)	0.83
P(PLANT | GIRL)	0.17
P(HUMAN | GIRL)	0.92
P(CAT | GIRL)	0.08
P(WOMAN | GIRL)	0.63
P(MAN | GIRL)	0.37

Table 12 Results by applying clustering on exemplars

We also tried to cluster exemplars obtained by Google with clustering package in WEKA [weka]. When calculating P(HUMAN | GIRL) and P (CAT | GIRL), we build a model with one cluster from HUMAN class and one cluster from CAT class, and use one cluster from GIRL class for the classification. Class HUMAN has 5 clusters, class CAT has 3 and Class GIRL has 3, so totally the above process is performed 45 times and the results show that one cluster from class CAT will give a desired result when used with clusters from other class. But this cluster is not the largest among the three clusters in CAT. Taking the largest cluster does not yield good results this time. These limited experiments indicate that clustering of text exemplars seems promising in resolving the "cat and dog" problem, provided we find a way to identify the right clusters.

Other experiments we carried out include adjusting the number of exemplars used and adding descriptive property of a class into a search query when collecting exemplars. Table 13 is a comparison of results obtained by specifying different numbers of exemplars used by the system. These exemplars are processed at a sentence level.

Conditional Probability	*Using first 50 exemplars*	*Using first 100 exemplars*	*Using first 200 exemplars*
P(ANIMAL | GIRL)	0.66	0.53	0.77
P(PLANT | GIRL)	0.34	0.47	0.23
P(HUMAN | GIRL)	0.86	0.56	0.43
P(CAT | GIRL)	0.01	0.15	0.01
P(DOG | GIRL)	0.13	0.29	0.56
P(PYCNOGONID | GIRL)	0	0	0
P(MAN | GIRL)	0.02	0.03	0
P(WOMAN | GIRL)	0.98	0.97	1

Table 13 Comparison between different numbers of exemplars (keyword sentence)

With 200 exemplars for each concept, the system gives the best result for LEVEL1 (ANIMAL and PLANT) and Level 3 (MAN and WOMAN), but not LEVEL2. With 50 exemplars, the system gives the best result for LEVEL2 but not the others. Using 50 or 100 exemplars the system also gives a correct overall mapping. Considering other experiments, using the first 50 or 100 exemplars processed at a sentence level seems to be a safe setting for the system.

We also tried queries augmented with class properties of each concept to find out whether this would give good results. Table 14 shows these queries. We let the system collect 100 exemplars for each concept based on the augmented queries and perform the same mapping process as described in Experiment (1) and (2). The results of these experiments are shown in Table 15 and 16. Different processing methods are compared.

Concepts	**Queries**
living+things	Living+things
animal	Living+things+animal+***Animalia***
plant	Living+things+plant+***Plantae***
cat	Living+things+animal+***Animalia***+cat+***Felidae***
human	Living+things+animal+***Animalia***+human+***intelligent***
man	Living+things+animal+***Animalia***+human+***intelligent***+man+***male***
woman	Living+things+animal+***Animalia***+human+***intelligent***+woman+***female***
tree	Living+things+plant+***Plantae***+tree
grass	Living+things+plant+***Plantae***+grass
frutex	Living+things+plant+***Plantae***+tree+Frutex
arbor	Living+things+plant+***Plantae***+tree+arbor

Table 14 Queries augmented with class properties

Conditional Probability	***Whole***	***Keyword Sentences***
P(MAN | HUMAN)	0.91	0.93
P(WOMAN | HUMAN)	0.09	0.07

Table 15 Experiment (2) Queries augmented with class properties

Conditional Probability	***Whole***	***Keyword Sentences***
P(ANIMAL | GIRL)	0.9	0.83
P(PLANT | GIRL)	0.1	0.17
P(HUMAN | GIRL)	0.78	0.83
P(CAT | GIRL)	0.22	0.17
P(MAN | GIRL)	0.14	0.16
P(WOMAN | GIRL)	0.86	0.84

Table 16 Experiment (2) Queries augmented with class properties

From the results, we can see that the method of using augmented queries misses expectations for Experiment (1) totally, but gives very good results for Experiment (2). Even using the whole processed document as an exemplar, the results were still very good. We would like to believe including class properties into queries is helpful. But the reason why sometimes this method does not give good results should be further researched.

Based on all these experiment results, we can see using 50 or 100 exemplars processed at a sentence level is generally a good setting for our mapping system. Text clustering on the exemplars will definitely further improve the exemplars' quality. Using queries augmented with class properties should help too.

5. Discussions

Our approach for ontology mapping employs naïve Bayes text classification method to calculate the conditional probability of one concept in Onto$_A$ given another concept in

Onto_B, which is used as an initial evidence of similarity and can be used in further process of the mapping [DPP05 and DPPY05]. The text exemplars used in this approach are obtained by search the web using semantic information found in the ontology definition file. Our experiment on the one hand produced positive supporting evidence for this approach, and at the same time have also revealed several limitations of this approach as well as issues that need to be further studied.

5.1 A web page is not a sample of a concept

When calculating a conditional probability, for example, P(MAN | HUMAN), samples of HUMAN in a given sample space should be collected and the conditional probability will be the ratio of MAN samples among HUMAN samples. When using a text classifier to calculate conditional probability of such two concepts, we can never get an individual sample of MAN or HUMAN, we can at the best get some strings that describe such a sample. Then conditional probability is estimated by counting words frequency and applying Bayes rule. This is totally different from the original definition of conditional probability of two concepts. Though sometimes we can find effective methods to make a better estimation, but that can never be accurate.

5.2 Popularity does not equal relevancy

In principle, we want to search for web pages that are highly relevant to the concept in question. However, relevancy is quite a subjective word. When we say a document is relevant to a search query, what do we exactly mean? We know the main algorithm Google uses for ranking the retuned pages is PageRank™ [gl]. Simply speaking, when page A has a link to page B, that link will be counted as a vote for page B. If A is an "important" page considered by Google, the vote will worth more. Although there are other sophisticated text matching algorithms combined to calculate the rank of a page, if many "important" pages have links to page B, page B will certainly has more possibility to show up in the first a few pages of search result. So if a web page has more links to it, it is possible that it get ranked higher. This may make sense to a human user sometimes but not necessarily good for our purpose. For example, the first result returned by Google recently for query "living+things+animal+human+woman" is an article in a blog. The article is about animal rights, it has little to do with the concept woman. But it is ranked the highest in the search result, just because the blog is called "woman to woman", and probably a popular blog on the web. To a human user, this popular blog might be what she or he wants, but the text information contained in this document and quite a few others obtained like this are not very helpful to a text classification. They are ranked higher by Google according to Google's algorithm. However, even though a search engine is our best choice available, a search engine's algorithm is never perfect and does not always work well with our approach.

One possible solution to this problem is to use a large amount of exemplars, hoping the irrelevant documents ranked by the search engine such as the above would only be a small fraction. We believe text clustering may help in solving or lessening this problem.

5.3 Weight cannot be specified for words in a search query

To form a search query, the parser usually concatenates several words together to get better results as explained in previous sections. However, current web search technologies do not allow us to assign weights to different words in a search query, even though, in most cases, what we care most is the last one or two words in the query. For example, in the first ten search results returned by Google recently for the search query "living +

things + animal + human “, only one of which has the desired text information to be used as exemplars for the LIVING_THINGS ontology, which explains HUMAN in a zoology context. All the others are general categorical information about animals, where the word “human” appears one or a few times. The first result returned is a Yahoo! directory page which listed links to animal related sites, for example, photos, categorical information to different animal sites etc. For posting a search query like “living + things + animal + human “, what we really look for are web pages about “human” while the other words such as living things and animal in the query provide a context for “human”, we are not really interested in pages which are generally about “animal” or about “living things”. In other words, we would like the search to be conducted with emphasis on “human” or to be weighted heavily on “human”. Unfortunately, there is no ways available currently to let a search engine understand such a request. The only advanced query available is with or without a string, or a phrase. How to improve a search engine or find a better way to use existing engines to accommodate a request like ours is an interesting topic to research and should be in our future work.

5.4 Relevancy does not equal to similarity

A search engine will return documents related to a query. What we are looking for are documents that explain or use the concept in a proper context, which is only a subset of the related documents returned as shown in Figure 13.

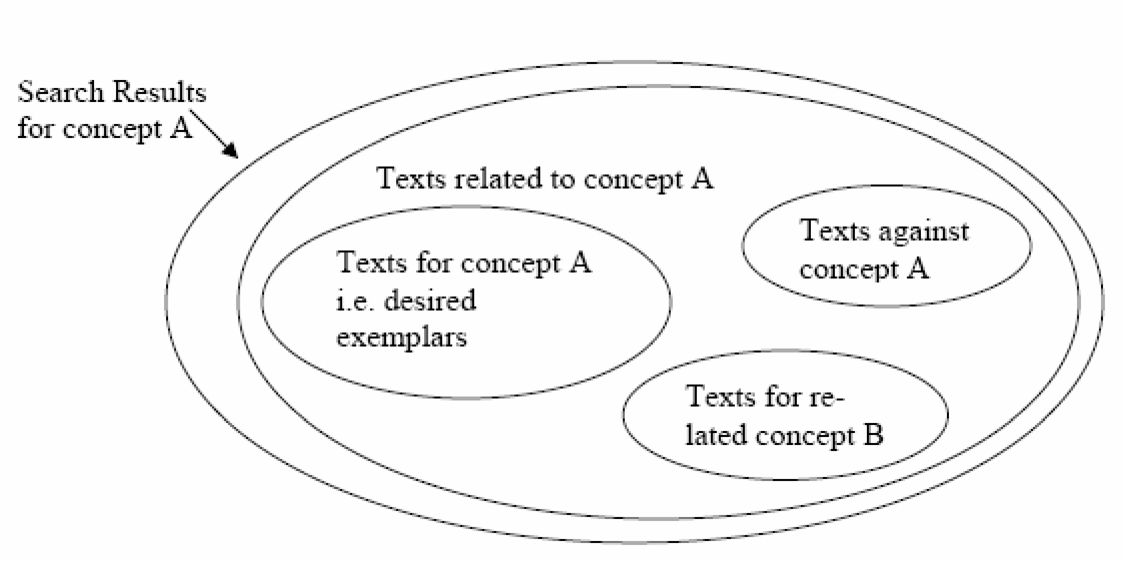


Figure 13 Relation of desired exemplars with different parts of search results

As a result, even though the returned search results very closely related to a query, it is still very difficult to correctly identify good exemplars. For example, web pages listed in Google's search results for query "WARPLANE+FIGHTER+PLANE" contain many text information for MISSILE. To be quantitative, 91,000 results are returned by Google recently for "WARPLANE+FIGHTER+PLANE", among which 40,400, nearly 45%, contain information for MISSILE. This agrees with our common sense, because a fighter plane has a close relation with missiles, for example, most fighter planes carry missiles and they can also be destroyed by missiles. However, in our experiments with WEAPONS ontology, this information causes our automated system to map FIGHTER-PLANE to MRBM, which is a class for medium-range missiles.

Neither search engine nor text classifier can differentiate these two types of information: one type is information about the concept in question and the other is information about things related to the concept. This makes it difficult to avoid wrong mappings in the results we obtained in WEAPONS ontology and explains further why we have the “cat” and “dog” problem with LIVING_THINGS ontology. Correctly selecting the desired exemplars out will improve the mapping results a lot. We believe that text clustering is a promising approach to achieve this goal and will find how to identify the right cluster in search results in our further research.

6. Related Work

Many people have used text classification methods to solve ontology mapping problem, but none has tried to automatically retrieve exemplars from the web for this purpose. Our work is motivated by OntoMapper [SPF02], a semi-automated ontology mapping tool, which is based on the text classification approach and also employs Rainbow as the classifier. Before mapping, exemplars for each concept need to be manually collected. The authors pointed out that the quality of mapping is greatly depended on the quality and quantity of exemplars.

CAIMEN [LG01] was developed to facilitate document retrieval and exchange among members of Community of Interests by mapping a user’s local ontology to the central document ontology shared by the community. The authors also used Rainbow for text classification. The exemplars are the documents provided by human users. A feature vector is calculated for each concept and a simple cosine similarity measure is applied on the feature vectors of a pair of concepts. A pre-selected threshold is used to decide a mapping. Like OntoMapper, CAIMEN is also only applicable if the users can supply exemplar documents.

To decide whether one concept A in Onto$_A$ is semantically identical or similar to another concept B in Onto$_B$, one has to have a similarity measure. GLUE [DMDDH02] uses the Jaccard coefficient [Ri79] similarity function, which is

$$P(A \cap B)/P(A \cup B)$$

$$= \frac{P(A,B)}{P(A,B) + P(A,\overline{B}) + P(\overline{A},B)}$$

So if A and B are identical, Similarity (A, B) should be 1 and if A and B are completely different, Similarity (A, B) should be 0 according to this measure. To train a classifier and perform classification, different learning techniques are used, one of which is a naïve Bayes text classifier. The exemplars and the full name of the classes are used to produce initial results. A full name is formed by names of every node on the path from the root class to the concept, which is similar to how our parser forms search queries. A meta-learner is developed to assign weights to results from different learners and calculate a final result as an input to the similarity function. Again, this system assumes the exemplars have been given and each text exemplar represents an individual instance of a class.

Some researchers in other applications also treat the WWW as a big sampling pool. For example, in [WPC05], the authors also use Google search results to estimate conditional probabilities. For a simple example, P(MAN | HUMAN) would be calculated as the ratio of the number of search results returned for keyword “man” and the number of search results for keyword “human”, which is 0.81 (1.83 billion divided by 2.25 billion found by Google recently). As we discussed in Section 5.4, conditional probabilities obtained in this way are in general very coarse.

7. Future Work

As mentioned in our discussion, if we were able to specify weights to search keywords, the quality of the exemplars obtained would be further improved. In other words, if we were able to search exemplars within some contexts, we would achieve better results. Actually, our current method creates some search context by augmenting the search query with ancestors of a concept, which is shown to be not very effective. For example, to search exemplars for class HUMAN in a LIVING_THINGS and ANIMAL context, we used query "living+things+animal+human", which gave us unexpectedly many documents about animal only with the word "human" appeared. If we cannot create such search context successfully, there may be ways to differentiate exemplars in different context afterwards, which can be done by text clustering. Text clustering will also help to identify the right group of exemplars shown in Figure 11 from the generally related documents effectively. Though we performed some preliminary text clustering experiments, yet we are still lack of a reasonable method to select the proper clusters, which leaves us some interesting future work.

Another direction for future research is to find a reasonable similarity measure. We are using conditional probability as a simple similarity measure to judge mapping performance, which is not always accurate, especially when applied to non-leaf classes. Because conditional probability measures how related two events are and two related events are not always necessarily identical. For example in one of our experiments with WEAPONS ontology, we have P(SELF-PROPELLED-ARTILLERY | APC) as the highest among the leaf classes of WeaponsA.n3 given APC. From a probability theory point of view, this is reasonable considering the close relation these two classes share. A good similarity measure will help produce more accurate mappings.

8. Conclusion

We proposed to automatically retrieve exemplars from the web for text classification based ontology mapping. We designed and implemented a fully automated system to collect exemplars and calculate conditional probability of two concepts as an initial similarity mapping. The tool can be very useful for ontology mapping tools and frameworks like [LG01, DMDDH02, SPF02, DPP05, and DPPY05] and other researches using such a conditional probability [WPC05].

Although the experiment results are mixed, they are in general encouraging and shed lights to the insight of this approach and further work. Two factors probably are most responsible for the less than ideal results. The first is the noise in the search results. Many keyword based search results are not really semantically relevant to the keywords. The second is that a search result is not really a random sampling of the web because all search engines return results according to their own ranking algorithms. How to address these problems and how to best utilize the imperfect exemplars in ontology mapping are some of the directions for future research.

References

[BLHL01] Berners-Lee, T., Hendler, J. and Lassila, O., The Semantic Web, *Scientific American*, 284(5), 2001

[cl] http://clusty.com

[DMDDH02] Doan, A., Madhavan, J., Dhamankar R., Domingos P., and Halevy A., Learning to Match Ontologies on the Semantic Web, *WWW2002*, May, 2002.

[DPP05] Ding, Z., Peng Y. and Pan R.: BayesOWL: Uncertainty Modeling in Semantic Web Ontologies", in *Soft Computing in Ontologies and Semantic Web*, Springer-Verlag, December 2005.

[DPPY05] Ding, Z,, Peng, Y., Pan, R., and Yu, Y., A Bayesian Methodology towards Automatic Ontology Mapping. *Proc of AAAI C&O-2005 Workshop*. 2005.

[i3c] http://www.atl.lmco.com/projects/ontology/i3con.html.

[gl] http://www.google.com/technology/

[kif] http://logic.stanford.edu/kif/kif.html

[Li04] Li, J., LOM a lexicon based ontology mapping tool, http://reliant.teknowledge.com/DAML/I3con.pdf.

[LG01] Lacher, M. and Groh, G., Facilitating the Exchange of Explicit Knowledge through Ontology Mappings, *Proc of the Fourteenth International FLAIRS conference*, 2001.

[MC96] McCallum, A., Bow: A toolkit for statistical language modeling, text retrieval, classification and clustering, http://www.cs.cmu.edu/~mccallum/bow 1996.

[Mi97] Mitchell, T., *Machine Learning*, McGraw Hill, 1997.

[NP03] Niles, I. and Pease, A., Mapping WordNet to the SUMO Ontology, *Proc of the IEEE International Knowledge Engineering conference*, 2003.

[Ri79] Rijsbergen, van C.J. *Information Retrieval. London*: Butterworths, 1979. Second Edition.

[RL02] Reed, L. and Lenat, D., Mapping Ontologies into Cyc. *Proc of AAAI* 2002.

[SPF02] Sushama, P., Peng, Y. and Finin, T., A Tool for Mapping between Two Ontologies Using Explicit Information, *AAMAS 2002 Workshop on Ontologies and Agent Systems*, 2002.

[sumo] http://www.ontologyportal.org

[UM06] Ushold, M. and Menzel, C., Achieving Semantic Interoperability & Integration Using RDF and OWL, http://cmenzel.org/w3c/SemanticInterop.html.

[weka] http://www.cs.waikato.ac.nz/~ml/

[wn] http://wordnet.princeton.edu

[WPC05] Wyatt, D., Philipose, M., and Choudhury, T., Unsupervised Activity Recognition Using Automatically Mined Common Sense. *Proceedings of AAAI-05*. pp. 21-27.

[WR04] Wiesman, F. and Roos, N., Domain Independent Learning of Ontology Mappings, *Proc of AAMAS* 2004.