Enhancing hypernym extraction for named entities using machine learning based classification

Bachelor's thesis

at the

Institute of Computer Science

at the

Faculty of Mathematics and Computer Science

of the

University of Heidelberg

by

Arndt Faulhaber

August 2005

Advisers:

Prof. Dr. Thomas Ludwig, Institute of Computer Science, University of Heidelberg

Dr. Rainer Malaka, European Media Laboratory GmbH, Heidelberg

Berenike Loos, European Media Laboratory GmbH, Heidelberg

Acknowledgments

My special thanks go to

Prof. Dr. Thomas Ludwig for accepting this thesis and the confidence put in me

Berenike Loos and Dr. Rainer Malaka for many fruitful discussions and a great supervision,

Robert Porzel for lending me his expert knowledge on how to write a scientific publication and numerous good ideas,

Hidir Aras for providing lots of information to start with,

Dorothee Reinhard and Pascal Hugelmann for the annotations,

Hans-Peter Zorn for many helpful hints on Java,

Nils Faulhaber, my brother, for being there, for all the encouragement and support,

Jens Teichert for the discussion about Support Vector Machines,

the creators of WEKA,

all those who contributed to make the Internet a rich source of helpful information like the Wikipedia project, LEO, Google™ etc.

and all others who contributed directly or indirectly to the creation of this document.

Table of Contents

Abstract

In this thesis I will discuss the construction and evaluation of a learning system that aims at extracting hypernyms for arbitrary named entities. The presented approach tries to accomplish this by using the web as a source of information. Namely querying Google^{TM} for a list of named entities and creating a corpus by retrieving the pages returned. From the corpus a set of snippets, i.e. sequences of words, is extracted that contains the named entity at the middle. A set of vectors is constructed, that is based on structural information conveyed by the words in the snippets and data extracted from the titles of the web pages. These vectors build the foundation for learning patterns that indicate a correct hypernym of the named entity. Only nouns are taken into account as hypernym candidates. To create a supervised learning scenario all extracted nouns are annotated as either representing a hypernym to a given named entity or not. Regarding these annotations the built system is evaluated. By comparing the results of the evaluation to those of a previously defined baseline method, the viability of the proposed approach can be assessed.

1 Introduction

Humans are used to perform different kinds of classifications and inferences nearly every second of their existence and often without even realizing it. These tasks range from looking at an object and classifying it as a house or a horse over listening to a new piece of music and correctly assigning its composer, e.g. Bach, to it. To learn taxonomic relations (a taxonomy is the classification of items into groups, for example dogs, cats and humans are part of the group of mammals; hence a taxonomic relation is a relation that denotes a relation between two such groups like a cat is a mammal or a mammal is a tetrapod, where "is a" indicates the taxonomic relation) from a written text in an area of knowledge that is new to the reader would be another example for the outstanding ability of humans to extend their knowledge by observing sources of information.

For computers to be able to perform similar tasks, at present, these problems pose major, if not unsolvable, difficulties. Though if computers were able to perform these tasks within a reasonable period, automating for example time-consuming literature lookups in research, invaluable resources could be saved and spent on more productive things than looking up a good set of textual sources.

This thesis will consider an aspect of one of the described classification problems, namely the extraction of pairs of words that are taxonomically related from a set of documents composed of natural text, and will try to analyze some possibilities of how computers might be used to automatically recognize such relations in a given set of documents. This thesis restricts itself to regard just a specific part of our language, named entities ("a named entity (NE) is a named object of interest like a person, organization, or location" as defined by NIST[1]), because standard corpora do not work for many of these, e.g. if the named entity did not exist before the creation of the corpus.

The goal of this work is to verify the feasibility of applying machine learning methods to the problem of finding hypernyms (a hypernym is a word that is more generic than a given word, for example "vehicle" is a hypernym of "car". Hypernyms also exist for named entities, for instance "country" is a hypernym of "Germany") for a given named entity. By evaluating the gain in precision in comparison to a baseline method, a quantitative testimony to the assumption that machine learning is a viable approach to tackling the introduced task will be given.

Why should the extraction of hypernyms help computers to support us in our everyday lives? The answer to this is relatively easy. To reasonably address problems

1 "National Institute of Standards and Technology", an official institution of the United States of America (http://www.itl.nist.gov/iaui/894.02/related_projects/muc/), last accessed August, 27 2005

that are not simply defined by a couple of formulas, as is the case with the above stated classification problems, one needs some knowledge of the world and how the principles interlock with each other in that world. Humans attain most of this knowledge by interacting with their surrounding. We learn for example that the sun rises in the east, or that trees normally grow outside and so on. Many of our decisions base on this common-sense knowledge. For machines to be able to perform similarly good decisions in a comparably flexible way, they are expected to need such kind of knowledge as well. Exactly at this point the goal of this thesis sets in. The process of hypernym extraction could be a step towards an automatic accumulation of knowledge and thus enable computers to expand their “view of the world” without the need of human intervention.

The approach taken in this work is to extract a set of features from the surrounding of a named entity (meaning e.g. 5 words in front and 5 words after the NE) and from the document it occurs in. In this information machine learning methods are employed to find similarities or patterns that indicate the existence of a taxonomic relation between the named entity and a word from its surroundings. Features can be structural information of the words surrounding a named entity, like the type of a word, the phrase it belongs to or the distance to the named entity. A feature related to the document as a whole is for example the information whether a word extracted as a hypernym candidate appears in the document's title.

The corpus for this work has been created from web pages returned by a query to Google^{TM}. A couple of reasons have been decisive for this choice. First of all, the world wide web is a vast resource of information. It not only contains encyclopedias like the famous Encyclopedia Britannica or the fast growing free encyclopedia named Wikipedia, it also contains a myriad of glossaries, scientific publications, essays, news articles – uncountable pieces of information, many of which are at the verge of being published at real-time (for example prices for shares or news). Following these arguments, one can assume that many systems will use the web as their source of information. Consequently, creating a corpus from a set of web pages for this evaluation seems to be an obvious choice and thus provides a realistic setup.

On the following pages a basic overview is given to create a picture of the context, in which this work can be seen. It starts with a description of natural language processing, being the general context of this thesis and proceeds to machine learning, being the more specific topic. To account for the scientific context of this work a small historic overview is added followed by a look at the state of the art.

1.1 Natural Language Processing (NLP)

Since the dawning of the information age the idea to use natural language in other ways, than just to speak, write or read, has emerged. In the beginning this idea was merely realized in telegraphs, morse codes and other ways of encoding. But later on, following the invention of modern electronic circuit based computers, the vision changed into the desire, that machines should learn to articulate themselves in the same way as humans do, even to understand what is said, and correctly interpret even irony or the way meaning can be altered by articulating parts of spoken language in a certain way and all these tasks should be performed artificially of course.

Shortly after the first achievements in computer based character recognition in the 1960s researchers of computer science and related subjects promised the creation of intelligent machines, as smart as any human, within a period of 20 years. Soon after though, the same researchers had to admit that the completion of this task was not as easy as initially perceived. Even at present, about 40 years later, the goal of creating machines with human-like intelligence still seems part of science fiction.

Machine-based language understanding is one of the classic areas of “artificial intelligence” and a subject of NLP. As common practice to all sciences, the problem (in this case machine based understanding of natural language) will be divided into small subproblems and these will be conquered step by step by the scientific community.

1.1.1 NLP - an overview

The main tasks in natural language processing are the following:

- ***Speech synthesis***: Is the term for artificial synthesis of human speech, also known as text-to-speech. This is probably the discipline featuring the longest and most obscure history with stories set in the height of the middle ages entwining around the so called “speaking heads”[2] (attributed to Gerbert of Aurillac (unknown-1003), Albertus Magnus (1198-1280), or Roger Bacon (1214-1294)). These stories are a testimonial that the idea of artificial creation of speech is not a new one albeit having lost none of it's fascination.
- ***Speech recognition***: Being more or less the counterpart to speech synthesis, and also known as speech-to-text, speech recognition is the name for techniques that will enable computers to convert human speech into a graphical representation, like letters and words. As such it is a much more recent idea than its counterpart, since computers are a relatively new invention.
- ***Natural language generation***: Not to be confused with speech synthesis, the goal of natural language generation is not the rendering from text into sonic waves, but the transformation of knowledge represented in a computer into human-understandable sentences.
- Machine translation: One of the driving forces behind NLP, in recent years, has been the motivation to develop a way to easily and accurately translate the many different languages in existence. Ideally, machine learning should become an adequate substitute for the fabled babel-fish from the book “Hitchhiker's Guide to the Galaxy” (Adams 1979).
- Question answering: Who does not know the situation: An interesting question comes up in a conversation but either the information does not (yet) exist or researching the answer poses too big an effort to be worthwhile, so the question remains unanswered and will probably be forgotten. For many questions answers do exist, but finding and reading the information is a time consuming process. Thus question answering aims at reducing the time needed to find an answer to a question and possibly giving an estimation for the quality of the returned answer.
- Information retrieval: Indexing even a smaller set of documents is an intricate task. With ever-growing anthologies, scientific publications for instance, finding a relevant set of documents poses a similarly growing problem. The discipline of information retrieval tries to find ways of choosing a best set of documents given a

2 See: Webster's Online Dictionary: http://www.websters-dictionary-online.org/definition/english/Sp/Speaking%20Heads.html, last accessed August, 28 2005

specific request.

- Information extraction: Information extraction (IE) is similar to the task of information retrieval just on a smaller scale. The parts retrieved are facts or pieces of information from a document (or small set of documents). The goal is to create structured data from an unstructured source of information, e.g. to collect dates of birth of persons.
- Text-proofing: We all know this particular part of NLP in its manifestation as automatic or semiautomatic spell checkers, grammar checkers or auto-corrections. Text-proofing tries to lessen the need for manual proof-reading.
- Language understanding: To understand the meaning of natural language, or parts of it, is the goal of language understanding. The problem about understanding is that often, especially when no information about the context is given, not even native speakers understand the meaning of a sentence. One just has to look into a book about quantum physics. Without intensive studies of that subject it will be hard to grasp the meaning of most of the content.
- Automatic Summarization: Fairly often only a fraction of a text holds information interesting to its reader or one only wants to get a picture of what issues have been covered by a text. In these cases a summary about the text under consideration would be of great use. Automatic summarization tries to extract the most important points of a text and hence providing an overview of topics addressed by the document.

In every single discipline listed the ***unknown word problem*** (the unknown word problem is said to occur, when one person (or entity) uses a word or term the other does not know) can and does arise, emphasizing the relevance of solutions for extracting knowledge about these unknown words (in this case named entities) to enable the systems to handle an out-of-vocabulary situation.

1.1.2 Machine learning - a central aspect in modern NLP

Machine learning (ML) can be defined as a way of how machines can generalize from experiences made (by presenting training data or interaction with their surrounding). The problem to be solved is generally perceived as a function that maps some input to a reaction or classification (f: X → Y). The goal of training is to approximate the real function f. That means a learning system that was presented some examples of houses, should learn the patterns, which all (or most) houses have in common. The function here could be the mapping of a pixel matrix into the domain {true, false}. These patterns should enable them to classify a new house, they have not "seen" before, as a house, by verifying the learned patterns. If the training is unsupervised, they might learn from patterns in the feedback from their environment. Often at the core of machine learning some kind of pattern recognition and pattern learning takes place.

Speaking of patterns leads more or less directly to natural language. In the early 20th century, A. A. Markov developed a probabilistic model known as ***Markov Model*** (Manning Schütze 1999). The first application of this model, in 1913, was modeling the letter sequence in Russian literature. Nowadays, ***Markov models*** are a commonly used representation method for machine learning based programs, especially in natural language processing. For example many POS-taggers (programs that assign each word of a sentence its corresponding part of speech, like noun, verb or adjective) make use of Hidden Markov Models (a slightly augmented version of the original Markov Model).

At the core of this thesis several different machine learning techniques are applied to recognize the sought for patterns (which point out taxonomic relations). To give a picture of the ML-methods put to work, these will shortly be described on the following pages.

The employed machine learning methods are based on three different basic techniques (some of the classifiers make use of more than one):

- Bayesian Networks
- Decision Trees
- Support Vector Machines (SVM)

Bayesian Networks are based on a simple theorem named after the English mathematician Thomas Bayes:

When A and B are events, the conditional probability of A assuming event B is

$$P(A|B) = \frac{P(B|A) * P(A)}{P(B)} \text{ , if } P(B) \neq 0.$$

Bayesian Networks are a way to represent dependencies of events in a compact way. It is an acyclic, directed graph with vertices representing events and edges representing direct influence from the source on the target. The vertices of a Bayesian Network additionally retain the probabilities for the represented event with all conditions for incoming edges. In Figure 1 that would mean vertex D retains P(D | A) and P(D | B) because D directly depends on events A and B. A, not depending on any other event would just retain P(A).

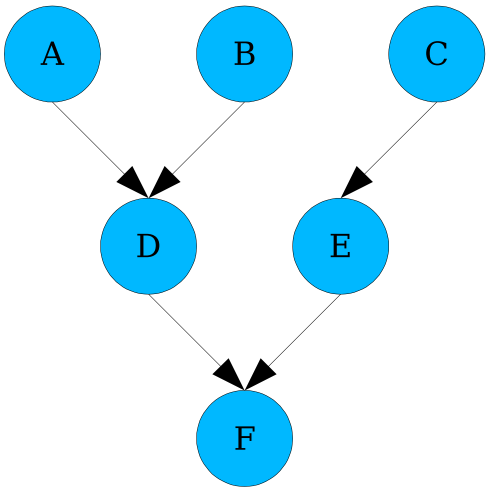

Figure 1: Bayesian Network, directed, acyclic graph

From the probabilities stored at the different vertices, one can now calculate other conditional probabilities. Given the probabilities:

- “the sun is shining”, P(SUN) = 0.5
- “I go climbing”, P(CLIMB) = 0.25
- “I go climbing when the sun is shining”, P(CLIMB | SUN) = 0.4

One now might ask what the probability was for the sun to shine, when I was climbing, P(SUN | CLIMB). Using Bayes theorem:

$$P(SUN|CLIMB) = \frac{P(CLIMB|SUN) * P(SUN)}{P(CLIMB)} = \frac{0.4 * 0.5}{0.25} = 0.8$$

A little calculation would return a probability for the sun to shine in case I was climbing. This example would fit into the graph of Figure 1, for instance, when C = SUN and E = CLIMB.

The goal of reaching a decision for a classification is accomplished by modeling features and their probabilities into such a network and testing the input's features against the network, the result with the highest probability will be returned as decision. An advantage of Bayesian Networks is, that they can return a probability for the decision reached, which might be a measure of how sure a decision is.

Decision Trees are a special representation of decision rules. The rules are nodes of a tree (the Decision Tree). Decisions are reached by traversing down the tree until a leaf of the tree has been reached. The leafs hold the resulting outcome.

Figure 2 models a Decision Tree for the “Monty-Hall-Problem”. In a show a candidate can win a car by choosing one of 3 doors. Behind 2 of them is a blank and behind the third is the car. The game works as follows:

- the candidate chooses an arbitrary door, without opening it immediately
- then the show master, who knows where the car is, opens one of the two others, bringing up a blank
- after opening the door the show master gives the candidate the option to change his decision

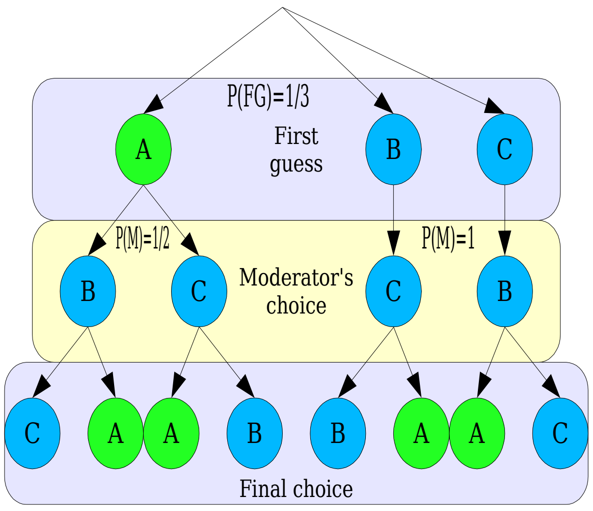

Figure 2: Monty-Hall-Problem, modeled as Decision Tree

The question that results is, what should the candidate do in order to maximize his chance to win. Modeling this problem as a Decision Tree is an easy way to reach a decision on what the candidate should do. From the Decision Tree the probability to win when the decision was changed and when it was not changed can be calculated. The probabilities are retrieved by multiplying the values of the edges starting at the root.

$$P(A|\text{changed}) = 2 * \frac{1}{3} * 1 = \frac{2}{3}$$

and

$$P(A|\text{unchanged}) = 2 * \frac{1}{3} * \frac{1}{2} = \frac{1}{3}$$

The goal for our problem at hand would be to construct a Decision Tree that will guide us to the correct decision with some features being nodes of the tree (for example if the distance from the named entity in question is 5 and the second word after the named entity is a verb followed by a determiner then it is a hypernym). Numerous algorithms exist to build a decision from sample data. The algorithms ***ID3*** and ***C4.5*** make use of entropy (a measure for information content (Shannon 1948)), while ***CART***, for instance, is based on squared probabilities of membership for each target category in the node. One of the advantages of Decision Trees is that such a tree is much easier to interpret than for example a Bayesian Network.

Finally, I want to present the idea behind Support Vector Machines (SVM).

The idea behind SVMs is to divide the space of positive and negative samples into two separate parts. One part should contain all positive samples and the other should contain all negative samples. The basic problem that has to be solved here, is that not all data can be linearly separated, for instance the XOR problem. That means, in two dimensions for example, no possible line exists that will separate the space $\mathbb{R}^2$ as described above, in case the positive samples are in the first and the third quadrant of a graph and the negative samples are in the second and fourth.

This problem can be solved, though, by transformation into a higher dimension. If we take the above example and project the vectors into $\mathbb{R}^3$ by setting the third dimension to zero for all vectors except for one. That one will receive anything else but zero, a hyperplane (actually in Euclidean space the plane, we know from geometry) does exist that separates the new space into two parts, each with only vectors of one type. After solving the XOR problem, the best hyperplane possible has to be found. The best possible hyperplane dividing the two spaces is chosen in that way, that the margin between the hyperplane and the nearest vectors (the support vectors, hence the name) to either side of it is maximized. The resulting hyperplane has the form:

$$h(\vec{x}) = \vec{w}^T \vec{x} - b = 0$$

The weights w and b have to be learned from the samples (as described above, margin maximization). Classification is performed by testing a vector against this hyperplane where:

$h(\vec{p}) \geqslant 0$ defines all positive vectors p and

$h(\vec{n}) < 0$ defines all negative vectors n.

1.1.3 State of the art

Significant progress has been made in the fields of machine based natural language understanding during the last two decades. Starting in the late 80's and during the 90's the MUC (Message Understanding Conferences) have played a major role in research standardization by putting together a set of corpora and tasks to be completed, employing these standardized corpora and hereby enabling the research community to compare their results. Since the late 90's one can notice a strong diversification of research areas associated with natural language processing.

One of the early approaches towards Information Extraction was the usage of handcrafted grammatical patterns to extract knowledge, like ***hyponym-hypernym*** relations (synonymous to taxonomic relations), from natural language texts (Hearst 1992). The main problem with pattern-based attempts at that time was the sparseness of data. That means these patterns appeared very seldom in common corpora. Modern applications of such patterns circumvent the sparse-data-problem by using the Internet as a source of information, demonstrated by Kilgarriff (2001), Evans (2003) and Cimiano (2004). Soon machine learning techniques were put to work to either learn patterns or to directly extract named entities from textual sources with an a priori set of patterns. To further enhance the precision of the analysis, a variety of methods have been applied including LSA (Latent Semantic Analysis, see Cederberg (2003)), integrating data repositories like WordNet (Feldbaum 1998) and gazetteers (a gazetteer is an index of geographical names), while further error reduction could be accomplished by combining multiple extraction approaches as shown by Florian (2003).

Using the Internet as a corpus yields an additional advantage vital for this thesis. It contains very recent information. News feeds and newspapers' websites present sources for a multitude of current news items. By means of a potent search engine this up-to-date information can be accessed in very short order. Thus superior accuracy is gained by processing named entities that only recently have been in the news. Give the example of a query about the name "Gerhard Schröder" within a political context. If an offline corpus had been crafted before Schröder was elected to be the German chancellor, one could hardly hope to extract the fact, that Schröder is the current chancellor. Whereas, taking into account topical information from the web, the query would at least make it possible that the result is the desired answer.

1.1.4 This approach

This thesis contributes mainly to the field of information extraction (IE). Being a closely related task, efforts in automatic question answering (QA) point into the same direction. The aspect of finding answers to inquiries known as definitional questions “What is XY?” aims at this problem, just from a different point of view. Additionally the problem at hand constrains XY to be a named entity of some arbitrary domain because more general concepts normally are not topical and thus can be extracted from standard corpora like BNC.

A number of approaches to extract hypernyms, without using a priori knowledge about a certain domain, exist. Evans, for instance, describes (Evans 2003) a system, called NERO, that is built on the retrieval of documents containing Hearst-patterns (see Section 2.3 for a detailed description of Hearst-patterns) by means of querying the Google™ search engine. In contrast the procedure presented in this thesis will not rely on Hearst-patterns alone (even though they will receive special attention) but will try to learn more general rules for inducing hypernymy relationships for named entities and thus is expected to yield a lower precision.

Another way to address the problem to find valid hypernyms for named entities is presented in Cimiano (2004). In this approach a set of queries, assembled by combining concepts from an ontology (a type of knowledge base) and a set of Hearst-patterns, is handed over to Google™ for retrieval. The classification then is based on the received numbers of hits for the (concept, instance, pattern)-triples. This method strongly differs from the one discussed in the following chapters, especially because the work at hand is not based on any given knowledge base.

Furthermore, most work has been put into analyzing extraction methods for English and therefore are biased to perform better when dealing with that language. This work concentrates on the German language which differs by a couple of major features, for example the possibility to create new compound words by concatenating two existing words (“Sonnen”[3] + “Haufen”[4] composes to a new valid word “Sonnenhaufen”[5]).

3 suns

4 heap / cluster

5 “cluster of suns”

2 Using machine learning techniques to improve hypernym extraction

This chapter will describe in detail the workflow behind the process of extracting hypernyms for a named entity and the ideas behind those procedures. First an overview with references to relevant sections is presented, followed by the sections describing each of the steps involved in building this system, from data acquisition to the selection of a best hypernym candidate.

2.1 The big picture

A coarse view on the pipeline shows five elements:

1. Data acquisition and preprocessing (described in Section 2.2)
2. Creating the training tuples (see Section 2.4.2)
3. Applying the baseline algorithm on the test data (Sections 2.3 and 3.3)
4. Using the classifier to assign a probability (or a true/false label) to a test vector (Section 2.5 and Section 3.4)
5. Evaluate the precision of the classifier (presented in Sections 2.6 and 3.4)

These main steps are split up into several smaller tasks.

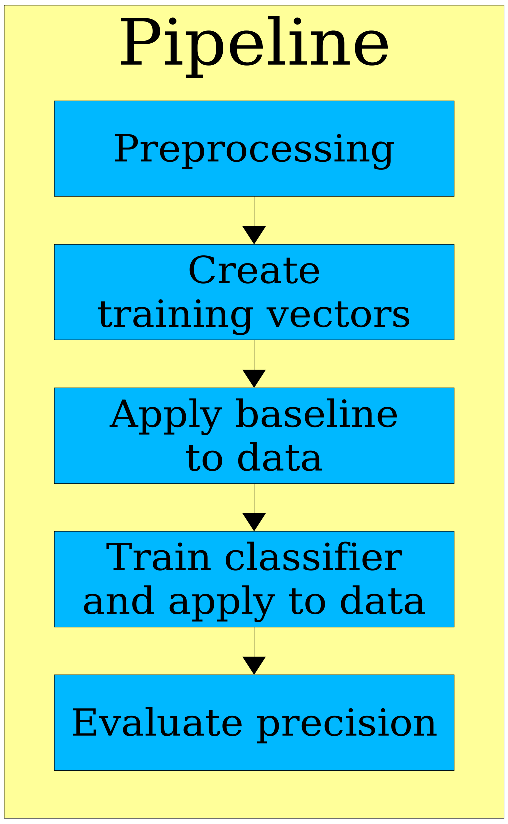

Figure 3: Coarse workflow

2.2 Preprocessing

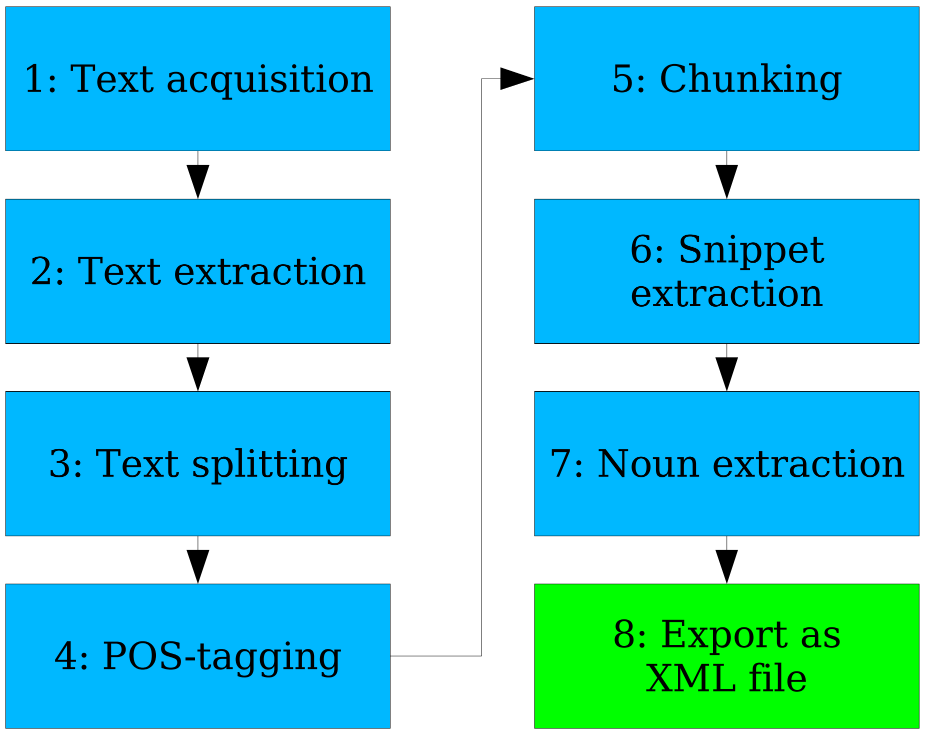

Figure 4: Preprocessing steps

Step 1: Text acquisition

As can be seen in Figure 2, the initial task is to access some textual source like a page from the world wide web or a file. This is normally done by sending a query to Google™ by means of the Google™-API and retrieving the search results. For the corpus that was composed for the evaluation in this thesis, with every named entity (except for "Weihnachten"[6]) a word conveying contextual information has been added to the search query, e.g. "Nike Mythologie"[7]. In addition to that the query was configured not to return PDF, PS or similar pages, that are not HTML and only pages in German should be returned (nonetheless, some English pages had been returned).

6 Christmas
7 mythology

Step 2: Text extraction

Depending on the source, the documents will contain artifacts that are not part of natural language. In the case of HTML pages these are HTML tags, menus and similar objects. Since we are mainly interested in the parts that are meant to form the text for the readers, those particles have to be filtered out. For parsing HTML different strategies exist. One of the easiest and quickest ones is to send the HTML source through a web browser and capture it's output. As a text based, non graphical browser, "***lynx***"[8] is meant to perform this task and is therefore herein put to work as a HTML filter.

Nevertheless, parts of the meta information might hold some interesting information. A title containing the named entity, for which a hypernym has to be retrieved, could be a hint towards the importance of the page regarding that specific named entity. This surplus information will also be extracted and stored for later processing during this part of preprocessing.

Step 3: Text splitting

After cleaning the text from all unwanted artifacts (or better most, since it is not a trivial decision, what to filter out), the resulting set of words, numbers and punctuation marks will have to be split into a series of particular tokens. Decisions have to be made whether a dot marks the end of a sentence or the end of an abbreviation. Wrong decisions on the meaning of such an item may lead to higher error rates in later steps of the processing.

One of the strategies applied by the system at hand is to use a list of common German abbreviations and expand them to their unshortened version.

8 http://lynx.browser.org/, last accessed August, 27 2005

Step 4: POS-tagging

In the fourth step this chain of words will be fed to a POS-tagger (POS stands for "***Part Of Speech***"), which assigns each of the tokens its corresponding part of speech tag. That means an adjective will be specified as ***adjective***, a determiner will be marked as ***determiner*** and so forth. Desirable would be a more detailed set of tags, for example with a separate tag for ***to be*** and its morphemes. The set distributed together with the ***TreeTagger*** (a POS-Tagger optimized for German and included in this project) is the STTS (Stuttgart-Tübingen Tagset) with 54 different tags as introduced in Schiller 1995. The precision of current POS-taggers applied to correct natural language averages around 95%. To get a picture of the error introduced by the POS-tagging, the integrated TreeTagger will be applied to an annotated corpus. An exertion to the last 1835 sentences of the ***Negra corpus*** (Brants 1999) yields a precision of 93.8%. In the table below an example of some tagged text has been supplied.

Token:[9]	***Man***	***sprach***	***Nike***	***verschiedene***	***Gegenstände***	***zu***
POS-tag:	PIS	VVFIN	NN	ADJA	NN	PTKVZ

Table 1: Example: POS-tagged text

Tag meanings:

PIS:	substituting indefinite pronoun
VVFIN:	finite verb
NN:	noun
ADJA:	attributive adjective
PTKVZ:	separated part of verb

9	"Man	sprach	Nike	verschiedene	Gegenstände	zu"
	"one	spoke	Nike	various	artefacts	to"

various artefacts were attributed to Nike

Step 5: Chunking

A chunker can then analyze the order of POS-tags and decide on phrase boundaries. An example for a chunk in German would be "verschiedene Gegenstände"[10] which should be marked as a noun phrase by the program (see Table 2 below).

Token:	***Man***	***sprach***	***Nike***	***verschiedene***	***Gegenstände***	***zu***
POS-tag:	PIS	VVFIN	NN	ADJA	NN	PTKVZ
Chunk tag:	B-NC	B-VC	B-NC	B-NC	I-NC	B-VC

Table 2: Example: Chunk tagged text

Tag meanings:

B-NC:	beginning of a noun phrase
B-VC:	beginning of a verb phrase
I-NC:	continuation of a noun phrase

A couple of other text enriching methods can easily be thought of and would surely help to increase the precision of the task at hand. But their implementation would exceed the time frame available for this thesis. Some of these enhancements will be discussed in the last chapter.

Step 6: Snippet extraction

The next step in the line of preprocessing is the search for relevant text snippets (a snippet in this context shall denote a small contiguous part of text) or sentences that contain the named entity in question. All tests that will be discussed in the course of this thesis used snippets of some defined radius with the named entity at the center position.

From now on a snippet will be understood as a vector of tokens (mainly words, numbers and punctuation marks) with their corresponding tags:

$S = <<token_1, tag_{11}, ... , tag_{1m}>, ..., <token_n, tag_{n1}, ... , tag_{nm}>>$

10 various items

In the case above and throughout this thesis the index m has a value of m = 2. However, further information can be added to the tokens (for example word stems, as is the case in the XML-example of Table 3), therefore a variable index has been chosen.

Step 7: Noun extraction

In step 7 the actual extraction of hypernym candidates takes place. All nouns (tokens with a POS-tag of "NN") surrounding the named entity will be put into their own data structure together with the snippets they have been found in. If a noun is found multiple times each snippet is added to the existing object.

For example a hypernym for "Nike" has to be extracted and the following two snippets have been found containing "Nike":

– "Man sprach Nike verschiedene Gegenstände zu"
– "Gegenstände mit Flügeln. Nike - Göttin des Sieges"[11]

From these snippets the words "Gegenstände" from the first and "Gegenstände", "Flügeln", "Göttin" and "Sieges" from the second would be extracted as nouns. Since "Gegenstände" occurs in both snippets both would be associated with this word, while each of the other words only are associated with one snippet, the one containing them.

11

Gegenstände	mit	Flügeln	.	Nike	-	Göttin	des	Sieges
artifacts	with	wings	.	Nike	-	goddess	of	victory

Step 8: Export as XML file

For performance reasons (caching) the gathered data is stored in XML format to avoid multiple preprocessing runs on the same data with the same processing parameters. A part of the XML output can be seen in Table 3, beneath.

```
<extractiongroup name='Lyra' frequency='2' >
  <hypernymset>
    <hypernym name='Lyra' frequency='2' />
  </hypernymset>
  <snippetset>
    <snippet size='21'>
      <token ID='0' name='Verehrern'>
        <tag ID='WORDSTEM'>Verehrer</tag>
        <tag ID='CHUNK'>I-NC</tag>
        <tag ID='POSTAG'>NN</tag>
      </token>
      <token ID='1' name='dieser'>
        <tag ID='WORDSTEM'>dies</tag>
        <tag ID='CHUNK'>B-NC</tag>
        <tag ID='POSTAG'>PDAT</tag>
      </token>
      <token ID='2' name='Statuen'>
        <tag ID='WORDSTEM'>Statue</tag>
        <tag ID='CHUNK'>I-NC</tag>
        <tag ID='POSTAG'>NN</tag>
      </token>
```

Table 3: Example: Part of XML format

2.3 The baseline

A baseline method is needed to evaluate results. Different types of baselines exist, majority class baselines, entropy based baselines (Porzel Malaka 2004) or a good (standard) approach. In this particular case the first two cannot be calculated, therefore a standard method will build the base to compare against. The following chapter details the baseline-related proposals.

Possibility 1:

Looking at natural language, we can observe certain patterns, that let us derive specific relations between words in a given sentence or part of a sentence. An example for such a pattern is the use of a term describing a named entity such as "Auerstein" directly before or after the named entity: "das Hotel Auerstein"[12] or "das Auerstein Hotel"[13]. In 1992 M.A. Hearst published a paper on the use of a set of lexico-semantic patterns to retrieve hyponyms from large text corpora (Hearst 1992) henceforth called Hearst-patterns:

(1)such NP as {NP,}* {(or|and)} NP
 Example: ... works by such authors as Herrick, Goldsmith, and Shakespeare.
 ⇒ hypernym("author", "Herrick")
 ⇒ hypernym("author", "Goldsmith")
 ⇒ hypernym("author", "Shakespeare")

(2)NP{, NP}*{,} {(and|or)} other NP
 Example: ... temples, treasuries, and other important civic buildings.
 ⇒ hypernym("building", "temple")
 ⇒ hypernym("building", "treasury")

(3)NP{,} {(including|especially)} {NP ,}* {(and|or)} NP
 Example: ... most European countries, especially France, England, and Spain.
 ⇒ hypernym("country", "France")
 ⇒ hypernym("country", "England")
 ⇒ hypernym("country", "Spain")

Table 4: Hearst-patterns from Hearst (1992) with slight modifications

12 the hotel Auerstein
13 the Auerstein hotel

Note that the above patterns do not only match the patterns meant to be Hearst-patterns, since they are context-free rules. As a result, e.g. (2) matches “NP other NP”, which is not a construct sought-after (or even correct), but the rules will match any of the patterns intended to. Analogous patterns exist for the German language (see appendix for patterns used in this project).

The problem with Hearst-patterns is, that the frequency of their appearance in corpora is generally very low. Resulting from the sparseness of Hearst-patterns the first attempt to create a baseline by just retrieving those patterns in the corpus did not work very well. During the search only for about 20% of the named entities examples of any of the patterns searched for had been found, supplying insufficient information to work with.

Possibility 2:

Looking at the corpus, the idea of terms being surrounded by others characterizing them, seemed to be a promising approach. So a new baseline was defined. This one searches snippets with the given named entity in their middle for nouns, counts them and assumes the most frequent of them to be a hypernym. Tests confirmed the assumption made and lead to a precision of roughly 40%, both for the training and test sets.

The first results that have been returned suggested two things:

- adding the frequencies of synonymous words like pub and bar might increase precision and
- in German some compound words are partly composed by a hypernym candidate. (like if the hypernym in question is “Göttin”[14] a possible compound word would be “Siegesgöttin”[15])

These thoughts lead to the formulation of the baseline's choice of a “best” proposition later defined in Section 2.6.1. Proof by experiment and the evaluation of overall precision can be found in Section 3.3.

14 goddess
15 goddess of victory

2.4 Learning to recognize taxonomic relations

The following part details the approach of applying machine learning algorithms to the hypernym classification problem. First, the problem of deciding on taxonomic relations will be transformed into a learning problem. As such it will be addressed by state of the art machine learning techniques. The structure of this chapter closely follows the procedure proposed by Mitchell in his book “Machine Learning” (pages 13 et sqq., (Mitchell 1997)), because it presents a well structured method for approaching machine learning problems and reflects all of the considerations needed for this section.

2.4.1 The learning experience

Before thinking about how to learn to automatically classify anything, detached from the question of what this anything will be, we have to ask from what experience the knowledge about the classification problem will be derived.

A couple of options stand at ones disposal, one of them is data that was annotated by an expert, another would be to learn from direct response to the system's own actions from the environment, as is the case with ***reinforcement learning***.

The chosen approach regarding the learning experience for the problem at hand can be summarized by the following four main points:

- The datasets have been extracted automatically by a part of the program.
- These have been manually annotated by experts
- and fed to the learner in a single run.
- Thereafter this configuration has not been changed in the course of data processing - the learning (or forgetting) has ended after the first presentation of examples.

This directly leads to the next question of how the datasets have been created and what the criteria for the annotation process were.

2.4.1.1 Acquisition and annotation of information

The acquisition of information to be manually classified has been done straight forward and is tightly connected with the baseline. All nouns surrounding the named entity in question were put in a list. From this list of hypernym candidates, HTML forms were automatically generated. Thus each of the annotators could choose all correct hypernyms.

Two annotation schemes have been constructed:

- A more liberal one: In this setting not only exact matches were marked as being correct, but also plural, genitive, accusative and dative forms, as well as words with obvious spelling errors.
- The second "more strict" set only used the correct forms and the words obviously in correct form with spelling errors.

The idea behind using a more generously evaluated version is to take more context information into account. Also some Hearst-patterns apply to the plural forms like "Hotels, wie z.B. Das Auerstein oder Zum Goldenen Adler"[16].

Some additional errors have been introduced by not looking at the snippet in which the hypernym candidate had been found and attach the rating to the snippet. Thus if two words appeared in a snippet where they do not indicate a taxonomic relation, this system would still interpret the example as one that indicates such a relation. For example if London is the named entity in question and city was indicated as correct hypernym during the annotation and a snippet like this will be found:

"... novel by Jack London. The story takes place in a city..."

To this system the snippet would erroneously be a valid indication that London is a city.

The additional excessive amount of time needed plainly was not at disposal considering an amount of about 14'000 snippets, that had been extracted, so this error had to be accepted.

16 hotels like e.g. "Das Auerstein" or "Zum Goldenen Adler"

2.4.1.2 The gold standard

That natural language is not exact and lots of misunderstandings do happen every day is an obvious fact. Therefore decisions made for an annotation by one person can be regarded as wrong by another. This dilemma and the need for the annotations to be as precise as possible, the best standard available, a gold standard, should be used to annotate the data. The error introduced by a weak annotation will strongly influence the quality of the training of the classifiers.

Initially introduced for diagnostic evaluations in a medical context gold standards have become a synonym for the best test method available. Ideally, a gold standard has a specificity of 100% (that means it does not state any false positives) and a sensitivity of also 100% (denoting all true positives will be indicated).

Needless to say ideal conditions rarely appear in a real setup. Thus for the annotation at hand the gold standard has been set by means of deriving the best choice of answers from the three annotations for the named entity at disposition. The three annotators discussed the decisions, that had not been made unanimously to fix down the final assessment. That is not just a majority vote, but every decision was made in total agreement.

2.4.2 The target function

Now that the learning experience has been defined, the following step in designing a learning system is to define a target function. A target function describes what kind of information is presented to a learning system and what the system should return. Like in checkers, we present a special board configuration and in return we want to know the best move we can make.

Two things have to be thought of:

- What is the input for the function? That input will actually be fed to the learning algorithms.
- What will the function return?

Being the easier part, the return values of the function will be defined first:

Ideally a classifier should return an exact answer to the question whether h_x is a hypernym of a named entity w, i.e. true or false. But in reality a variety of examples exist, where not even humans can decide on a correct answer when looking at a sequence of words. Reasons for this include:

- ***Lack of information:***
  If a document does not contain the named entity in question (an extreme case would be, if we queried for “mammal”, but only have a set of documents at hand that do not contain the word “mammal”).
- ***Ambiguity:***
  Some words do have two or more meanings, like “rock” for example which can be a stone or a type of music.
- ***Incorrectness of documents:***
  There is no such thing as: it is God-given, that every document in existence states the absolute truth. Documents can be misunderstood, misinterpreted, deliberately falsified or simply a little orthographic mistake can turn a bank into a tank or a trifle into a rifle.

Thus the data worked on can be of different qualities and therefore the output of a classification should be able to take heed of this fact. Assumed that the same mistakes generally happen less often than the correct usage of words, one can infer, that if a number of definitions for a named entity, whether direct or indirect, have been found, the one occurring the most should be the right one.

Taking the argumentation above as a starting point for defining an adequate output for the fuction, the most appealing output would be a rating $c_w(i) \in [0,1]$ for a given input vector $i(h_x)$ and a hypernym candidate h_x of w. It should provide a measure of how "good" a hypernym candidate h_x fulfills the criteria of being a hypernym of w.

More tricky is the construction of the input vector for the target function. It should be composed in such a way, that a classifier actually learns criteria indicating a taxonomic relationship and that a classification does not merely end up in some statistic noise or a rather complicated random choice function. Further it should be able to generalize and correctly classify samples it has not been trained on.

The major hypothesis this training problem is based on, is the supposition that the immediate environment of a named entity provides a closer description of this named entity. That means information about the surrounding has to be part of the input vector.

Words or even sentences in the named entities environment could be made part of the input, but that would lead to a vast space of input vectors – the German Duden lists more than 250'000 words and expressions, moreover it is very easy to create new words in German: One only has to take two nouns and puts them together and gets a syntactically (and often semantically) correct new word: "Stuhl"[17] + "Bein"[18] ⇒ "Stuhlbein"[19], "Stuhlbein" + "Farbe"[20] ⇒ "Stuhlbeinfarbe"[21] and so on. Supposing we use snippets consisting of the named entity, 10 words in front and 10 words behind the named entity and an amount of 250'000 different words. That would lead to an amount of $250'000^{20} + 1 \cong 10^{108}$ possible vectors (the 1 stands for the named entity in the middle, which is fix). In fact many combinations are complete non-sense like "... der der die das Triller der der das die ..."[22] and surely will never show up in a meaningful text. Trying to present the tokens of the snippets to a machine learning algorithm would need tremendous amounts of samples, implying a considerable amount of data to be annotated.

To circumvent the data sparseness problem (which would result in too few examples being fed to a classifier), and still to add considerable knowledge about the surrounding, some structural information about the neighboring words is added to the input, namely POS-tags and chunk tags (as introduced in section 2.2):

17 chair
18 leg
19 leg of a chair
20 color
21 color of a leg of a chair
22 "... the the the the warbler the the the the ..."

i = <POS-tag_-10, ..., POS-tag_-1, POS-tag_+1, ..., POS-tag_+10, Chunk-tag_-10, ..., Chunk-tag_+10>

The POS-tag of the named entity should either be NE (if the POS-tagger knows it to be a named entity) or a noun, hence we can spare this information, since nothing would be gained from it.

This creates an amount of $54^{20} * 10^{21} \cong 4.45 * 10^{55}$ different vectors, a tiny fraction of the above (about a 10^{52}-th). Again many of the composable vectors are very unlikely to appear in a text.

To relate a hypernym candidate to the context, another feature has been added to the input vector, the distance from the named entity (***WordDistance***). This distance is given as an integer value denoting the relative distance to the named entity evaluated. A value of -5 would indicate the fifth token in front of the named entity is the hypernym candidate while a value of 3 would mean the third token after the named entity.

i = <POS-tag_-10, ..., POS-tag_-1, POS-tag_+1, ..., POS-tag_+10, Chunk-tag_-10, ..., Chunk-tag_+10, ***WordDistance***>

In addition to this purely structural information another set of vectors, containing some knowledge about the document, have been fed to the classifiers:

- ***NE-in-Title***: A boolean value of whether the named entity occurs in the title of any of the web pages used as corpus.
- ***Hypernym-in-Title***: An analogous boolean about the presence of the hypernym candidate h_x appearing in the web pages' titles.
- ***Anaphora-hint***: A hint towards the possibility of an anaphora being used in the snippet (in German words like “dieser”, “diese”[23] etc.), again coded as boolean: such a word has been found between w and h_x or not.
- ***Hearst-pattern***: Are the named entity w and the hypernym candidate h_x part of a Hearst-pattern, again coded as a true/false value.

Finally, for a classifier to be trained a field with a classification has to be added, so the

23 this / that

learner can distinguish between correct samples and incorrect ones. All features combined result in a vector of the form:

i = <POS-tag_-10, ..., POS-tag_-1, POS-tag_+1, ..., POS-tag_+10, Chunk-tag_-10, ..., Chunk-tag_+10, WordDistance, Hypernym-in-Title, NE-in-title, Anaphora-hint, Hearst-pattern, Classification>

with a signature (here the signature lists the number of different values a field of a vector can hold) of:

s_i = <...20 times 54..., ...21 times 10..., 20, 2, 2, 2, 2, 2> (2.84* 10^{58} possibilities).

Put together this will result in a target function:

$$c_w(i(h_x)) \rightarrow [0, 1]$$

to be learned by a classifier.

Note that the hypernym candidate is not explicitly part of c_w, but implicitly coded into it by means of the value of ***WordDistance***, encoding the distance from the named entity in the middle.

2.4.3 Representation and learning algorithms

The representation of the data strongly depends on the chosen learning algorithm. Various learning algorithms have been used to evaluate their performance for the given problem, ranging from Bayes networks over Decision Trees to Support Vector Machines. Each with it's own representation of data.

While Decision Trees belong to those methods that represent data in symbolic form, for which the representations can easily be interpreted, Bayesian Networks and Support Vector Machines belong to the class using a subsymbolic representation (like probabilities or polynomials).

By employing WEKA, a framework containing multiple classifiers, many different ML-methods can be tested. The final choice of classifiers, consists of those listed below (many others have been too slow, so that no results could be attained during the evaluation phase, especially multilayer perceptrons took a long time to be trained):

- AODE (Averaged One-Dependence Estimators (Webb 2002)), a Bayes based approach, averaging over all of a small space of alternative naive-Bayes-like models
- ADTree (Alternating Decision Tree (Freund 1999)), a generalization of Decision Trees, voted Decision Trees and voted decision stumps
- J48, C4.5 based Decision Tree (Quinlan 1993)
- NBTree, a Decision Tree with Naive Bayes classifiers at the leaves (Kohavi 1996)
- SMO, a Support Vector Machine with Sequential Minimal Optimization (Platt 1998)

For the first three of these, different parameters have been examined, for example, the number of boosting iterations executed by the Alternating Decision Tree classifier.

2.5 The classification process

Data has been extracted, a function has been defined that should be learned and the machine learning algorithms are chosen. The next step in the processing sequence is the application of the machine learning algorithms. This process consists of two steps, training and classification. For these the set of assembled vectors, described in Section 2.3, is divided into two parts, a training set and a test set.

To train a classifier, the set of training vectors is presented to the classifier. Depending on the method used, the presented set in turn will be subdivided into two more sets. Set one is used to approximate the function resembled by it's vectors, and the other set, called validation set, is used to test the learned function to avoid overfitting (overfitting happens when a classifier adapts too closely to the function resembled by the training vectors, thereby loosing the desired ability to generalize, i.e. to correctly classify unknown examples). In the case of overfitting decision trees, for instance, often employ a technique called pruning, which is the term for cutting off branches from the tree and thereby reducing the specificity of the tree. The methods used to avoid overfitting strongly depends on the type of classifier at disposal.

At this stage, after the training, the construction of the learning system is finished, thus, the system is ready to classify unknown vectors. For the task at hand, rating the vectors from the test set actually means assigning each occurrence of a hypernym candidate a rating that denotes how well it suits the learned patterns. A good rating (close to 1) indicates a hypernym candidate as a good choice to represent a correct hypernym.

2.6 Selection functions

After every single occurrence of any hypernym candidate has been assessed by a classifier or by the baseline, a choice has to be made, which of all the hypernym candidates is the best (second best, third best etc.). This can be as easy as choosing the elements of one of two equivalence classes as would be the case using a Support Vector Machine (SVM), that assigns true/false values to each item. Nonetheless the judgment can get more complicated, if a more diverse set of ratings has been assigned to the hypernym candidates.

First, an outlook on prevalent selection functions will be given, followed by a description of the functions applied during the evaluation in the next chapter. While a separate function will be introduced specifically for the baseline, the sections on unweighted and weighted functions only regard the choosing of a “best” item of a set of items that have been considered by a classifier.

2.6.1 A selection function for the baseline

With the notions from Section 2.3, where the ideas behind the baseline have been introduced, in mind, the method by which the “best” hypernym candidate (out of all extracted hypernym candidates) will be chosen, will be defined. First I will present a formalism to merge groups of synonyms and similar compound words. Then a function is defined to choose a “best” candidate, taking into account the available information, i.e. the group a hypernym candidate belongs to and its frequency of occurrence.

Let w be an named entity for which a hypernym has to be returned, let H_w be the set of all extracted hypernym candidates for w (all tokens surrounding w in any of the extracted snippets (as described above) with a POS-tag of “NN”, marking them as nouns), let P be the set of possible partitions on H_w, and let $P_0 = \{\{h_0\}, \{h_1\}, ..., \{h_n\}\}$ be an initial partition of H_w.

Furthermore let a and b be two arbitrary words, whereas syn(a, b) is defined as a binary predicate that is true, if and only if the two words a and b are synonyms.

Then a ***grouping*** function

$$f: H_w \times H_w \times P \rightarrow P$$

shall be defined for $h_x \in H_x$, $h_y \in H_x$ and $P_m \in P$ as

$$f(h_x, h_y, P_m) = \begin{cases} ((P_m \backslash H_x) \backslash H_y) \cup \{H_x \cup H_y\}, & \text{if } syn(h_x, h_y) \\ P_m & \text{, otherwise} \end{cases}$$

The function f creates a new partition based on P_m by merging two elements $H_x \in P_m$ and $H_y \in P_m$, if $h_x \in H_x$ and $h_y \in H_y$ are synonyms known to the system, and leaving the other elements unchanged.

The process of merging sets containing synonyms shall be referred to by the name of "***syngrouping***".

Let again a and b be two arbitrary strings, whereas se(a, b) defines a binary predicate that is true, if and only if either a begins with b or a ends with b (in the later examinations the predicate se(a, b) will always be used without distinguishing upper case from lower case characters).

Another ***grouping*** function

$$g: H_w \times H_w \times P \rightarrow P$$

is then defined for $h_x \in H_x$, $h_y \in H_x$ and $P_m \in P$ as

$$g(h_x, h_y, P_m) = \begin{cases} ((P_m \backslash H_x) \backslash H_y) \cup \{H_x \cup H_y\} & \text{, if } se(h_x, h_y) \\ P_m & \text{, otherwise} \end{cases}$$

This fonction, hereafter referred to as "***compgrouping***", creates a new partition based on P_m by uniting two elements $H_x \in P_m$ and $H_y \in P_m$, if a hypernym candidate $h_x \in H_x$ starts or ends with hypernym candidate $h_y \in H_y$ and leaves the other elements unchanged.

A new partition P_f is created by applying f for any pair h_x and h_y and the partition P_m attained by the last application of f ($P_{m+1} = f(h_x, h_y, P_m)$), beginning at P_0. Subsequently P_{fg} is computed by applying g to each pair h_x and h_y and the partition P_m attained by the last application of g ($P_{m+1} = g(h_x, h_y, P_m)$), beginning at P_f.

In the following the proof that P_{fg} is still a partition will be shown:

> Supposing an element h_x exists with $h_x \in H_k$ and $h_x \in H_l$, $H_k \neq H_l$. This is an impossible condition at the construction of P_0 and cannot happen after any application of f or g, since only unifications occur. Furthermore all elements $h_x \in H_w$ are still in P_{fg}, because sets will only be unified by either f or g, hence no elements are extracted from any set and so P_{fg} must be a partition.

The function $occ(h_x)$ returns the number of occurrences of a hypernym candidate h_x in the given set of snippets.

Each element P_j of a partition P is assigned a value by the function $met(P_j)$ that is the sum of all occurrences $occ(h_x)$ of each hypernym candidate $h_x \in P_j$ in P_j.

$$\mathrm{met}(P_j) = \sum_{h_x \in P_j} \mathrm{occ}(h_x)$$

The best selection $best_{base}(H)$ is defined as the most frequent hypernym candidate $h_s \in P_{max}$ in the set $P_{max} \in P$ with the highest assigned value of any $h_x \in P_{max}$:

$$P_{max} = \arg\max_{P_j} (\mathrm{met}(P_j))$$

$$\boxed{h_s = best_{base}(H) = \arg\max_{h_x} (\mathrm{occ}(h_x))}$$

Accordingly, the second best would be the most frequent hypernym candidate in the subset with the second highest value ($best_{base,2} = best_{base}(H \setminus P_{max})$) and so forth.

One could ask why the selection function does not choose another frequently occurring hypernym candidate from the same group P_{max}. The answer to this question is, that the aim of grouping synonyms and similar words together is to create a group of words, that represent one and the same meaning, so any one sample out of a group is meant to fully represent the group it was taken from. Hence to choose a hypernym candidate with a different sense has to be gathered from another group of words.

For later evaluation of the baseline, a function ***BaseN*** is defined, which returns 1 if any of the N best-rated (according to $\text{best}_{\text{base}}$) hypernym candidates has been annotated as a correct hypernym within the used annotation.

2.6.2 Group-insensitive functions

One of the more simple selection functions is to search for the hypernym candidate, that was attributed the highest rating of all hypernym candidates by a classifier. I have implemented this selection function and will refer it as ***BestNMax***, where N is a substitute for the number of best-rated hypernym candidates being regarded. That means Best3Max would return the 3 hypernym candidates with the highest ratings assigned to them.

Formally ***BestNMax*** can be described as follows:

Since one word that is a hypernym candidate can occur multiple times in the set of extracted snippets and be assigned different values, we have to look at the set K of all occurrences of hypernym candidate words. Two occurrences k_i, $k_j \in K$ can both refer to the same hypernym candidate, but can be assigned different classification values $c_w(k_i)$, $c_w(k_j)$. We assume all classification ratings $c_w(k_x) \in [0, 1]$.

The best choice $k_m = \text{best}_{\text{max}}(K, w)$ will be the hypernym candidate with the highest assigned rating $c_w(k_m)$, for $k \in K$:

$$k_m = \text{best}_{\text{max}}(K, w) = \arg\max_k (c_w(k))$$

The second best then would be $k_{m,2} = \text{best}_{\text{max}}(K \setminus \{k_m\}, w)$ and so forth.

2.6.3 Group-sensitive functions

Group-sensitive means, that the frequency of occurrence of a hypernym candidate h_x and the size met(P_j) of the partition P_j it belongs to, have an influence on the evaluation of that hypernym candidate. Looking at the evaluated data, taking into account the amount of times a hypernym candidate has been come across in the set of snippets, can increase the accuracy of classification. In accordance to that observation a second selection function has been included in the system, henceforth called ***BestN***. Again, N denotes the amount of hypernym candidates returned.

The design aspects for ***BestN*** are:

- The resulting rating of a hypernym candidate should be element of the interval [0, 1].
- A higher number of occurrences should increase the new rating.
- All ratings assigned by a classifier should be taken heed of, where high values should result in a better value for the recalculated overall rating for a hypernym candidate.

Hereafter, ***BestN*** being a simple function that fulfills these criteria will be introduced. ***BestN*** is based on a rating $r(H_k)$ for the elements H_k of the partition P_{fg} of H as described in Section 2.6.1 The rating $r(H_k)$ for an element $H_k \in P_{fg}$ is calculated by summing up the occurrences $occ(h_i)$ of all hypernym candidates $h_i \in H_k$ in H_k.
Let $r_{max} = max(r(H_k))$ be the maximum rating of any set H_k of P_{fg}. And let K be the set of all occurrences of extracted hypernym candidates k, as introduced in the preceding section (2.6.2). The normalized rating $c_{norm}(h_x)$ for a hypernym candidate h_x is received by dividing the sum of all ratings $c_w(k)$ for any $k \in K$ in K, referencing the same hypernym candidate h_x, by r_{max}. Let $R \subseteq K$ be the set of all $k \in K$, that reference the same hypernym candidate h_x. Then:

$$r(H_k) = \sum_{h_i \in H_k} occ(h_i)$$

$$r_{max} = max(r(H_k \in P_{fg}))$$

$$c_{norm}(h_x)=\frac{(\sum_{k\in R} c_w(k))}{r_{max}}$$

The weighting function ***BestN*** chooses the hypernym candidate $h_m \in H_b$ with the highest normalized rating c_{norm} from the highest rated set H_b as the best proposition best(H). The second best $h_{m,2}$ = best(H \ H_k) is the item with the highest c_{norm} chosen from the set with the second highest rating and so forth.

$$H_b=\arg\max_{H_k}(r(H_k))$$

For $h_x \in H_b$ we obtain

$$\boxed{h_m=best(H)=\arg\max_{h_x}(c_{norm}(h_x))}$$

3 Evaluation

The following chapter will present the results of the evaluations performed to support the previously positioned hypotheses.

- The defined grouping functions (***syngroupng*** and ***compgrouping***) increse the performance of the baseline.
- Machine learning is an approach that supersedes the performance of the baseline

The evaluation will start with an analysis of the baseline method and its enhancements, continue to the numerous aspects of machine learning, i.e. the configuration presented to the classifiers, and finally proceed to the presentation of the numbers achieved by the best-performing ML-based method and compare the result with the baseline's precision. This will show whether the method presented in this thesis represents a viable approach to solve the discussed problem of identifying hypernyms for named entities.

3.1 The evaluation setup

For the evaluation 117 named entities had been chosen more or less randomly by a third person not involved with this thesis. These can be categorized into two major classes:

- A set picked from a specific domain consisting of localities like hotels and restaurants contributing 70 of the 117 named entities and
- a set of named entities from miscellaneous domains constituting the missing 47 named entities.

The complete list can be found in Appendix 6.3.

The set of named entities has been divided into a training set of 67 named entities and a test set of 50 named entities with a homogeneous distribution of the two classes. Those sets have been carefully crafted to deliver as near as the same performance when classified by the baseline (based on the values reached using the liberal annotation). This means that the percentage of correctly extracted hypernyms for the named entities of the training set resembles the result for the test set.

3.1.1 Choosing the best proposition

In Section 2.6 three different functions have been introduced for choosing the N best proposals from a set of hypernym candidates. All of the following evaluations have been done using nine different variations based on these three selection functions.

In the following, a correctly returned hypernym candidate will indicate an exact match to one of the terms marked as a correct hypernym to a given named entity. If for example, in the annotation used, only “Hotel” is listed as correct hypernym for the named entity “Auerstein”, any hypernym candidate returned but “Hotel” will result in a failure of the test. This even extends to case sensitivity (“hotel” ≠ “Hotel”).

- ***Base1***, ***Base2*** and ***Base3*** choose the best rated hypernym candidate from the best N rated groups of hypernym candidates. A test using BaseN will succeed if the chosen hypernym candidates actually contain a hypernym candidate for the named entity.
- ***Best1***, ***Best2*** and ***Best3*** return the best rated hypernym candidate for each of the N (N=1, 2, 3 respectively) best rated groups of hypernym candidates. This test will succeed if one of the N selected hypernym candidates is a correct hypernym.
- ***Best1Max***, ***Best2Max*** and ***Best3Max*** delivers the N (N=1, 2, 3 respectively) hypernym candidates having received the highest ratings during classification. A positive result is reached, if one of the hypernym candidates decided on complies to the required taxonomic relation.

The higher N-values have been inspired by the possibility of adding a decision layer after this classification process. Based on additional knowledge, like contextual information or an ontology containing a class returned, this decision layer might choose an appropriate hypernym candidate out of a set of answers, or give the user a selection of propositions in the case of a dialog system. Furthermore the additional results should provide a picture of the distribution of accurate results among the highest-rated hypernym candidates.

3.2 Methodology

The formulas commonly used for analysis of the quality of NLP techniques are ***precision*** and ***recall***. Precision is the measure of how many of the suggested results are correct (relative frequency of correct occurrences in the result set). Recall denotes the percentage of accurate answers that have been found in the analyzed corpus. While precision can be computed in any IE setting, recall can only be computed when using an unchanging corpus. That means when mining the web using Google™ for example, we cannot realistically compute the value for recall, because our corpus is actually a large fraction of the world wide web (currently something over 8 billion web pages), though we can evaluate the recall value taking into account the documents that really have been processed (after having the search engine do a specific preselection based on our search query). For direct comparison another measure has been introduced, the ***F-measure***, namely the harmonic mean of precision and recall (as proposed by van Rijsbergen (1979)).

Formally precision, recall and F-measure are defined as follows:

Let S be the set of extracted elements, $C \subseteq S$ the subset of correctly extracted elements in S and $A \supseteq C$ the set of all correct elements in the corpus, then

precision $$P = \frac{|C|}{|S|} \quad ,$$

recall $$R = \frac{|C|}{|A|} \quad \text{and}$$

F-measure $$F = \frac{2 * P * R}{(P + R)} \quad .$$

As already mentioned above, calculating the recall value can be tricky, especially so, if an exact definition of the set of correct candidates for an extraction is hard to determine. This is the case, when the domain of terms to be extracted is left open. For instance in the domain of music rock is a more or less clearly defined term, but leaving the domain open, a multitude of different meanings can be correct, especially in a metaphorical sense. Looking at “You are my bridge over troubled waters”[24], “you” specifies a person and hence this means the person is a bridge. Because this is one of the patterns used in this thesis, that “correctly” implies a meaning for an entity (in this case “you” is a substitute for a named entity and could easily be exchanged by “Mark Twain is my bridge...”). Thus we cannot mark it as an invalid example, and therefore a strict definition is hard to set. Taking these difficulties into account, in the following evaluation only precision will be regarded.

24 Simon and Garfunkel

3.3 Quantitative analysis of the baseline

In the introduction of the baseline (Section 2.3) two hypotheses have been proposed towards an enhancement of the baseline.

First, merging synonyms together enhances the precision of the baseline, introduced as "***syngrouping***".

Second, merging compound terms starting or ending with another hypernym candidate, introduced as "***compgrouping***".

The two tables below, Tables 5 and 6, show the performance of the baseline without either of the grouping enhancements. The first one is based on the strict and the second one on the liberal annotation.

	Base1		Base2		Base3	
	Training set	Test set	Training set	Test set	Training set	Test set
Localities	14/40 (35%)	11/30 (37%)	20/40 (50%)	17/30 (57%)	23/40 (58%)	18/30 (60%)
Misc. domains	5/27 (19%)	5/20 (25%)	8/27 (30%)	10/20 (50%)	8/27 (30%)	10/20 (50%)
Sum	35/117 (30%)		55/117 (47%)		59/117 (50%)	

Table 5: Reduced baseline precision without grouping, strict annotation

	Base1		Base2		Base3	
	Training set	Test set	Training set	Test set	Training set	Test set
Localities	16/40 (40%)	11/30 (37%)	21/40 (53%)	18/30 (60%)	24/40 (60%)	19/30 (63%)
Misc. domains	7/27 (26%)	5/20 (25%)	11/27 (41%)	11/20 (55%)	11/27 (41%)	12/20 (60%)
Sum	39/117 (33%)		61/117 (52%)		66/117 (56%)	

Table 6: Reduced baseline precision without grouping, liberal annotation

Obviously, the strict annotation reduces the performance of the baseline. This had to be expected since only the set of correct answers has been reduced while the selected hypernym candidates stayed the same. Another palpable aspect that can be observed is that the precision regarding the localities, with either type of annotation, is much higher than the precision measured for the named entities of the set consisting of miscellaneous domains. 36% for Base1 with strict annotation and 39% with liberal annotation of the localities have returned a correct result. In comparison to only 21% and 26% correspondingly of the hypernym candidates from the miscellaneous domain set.

This difference should be kept in mind and will be regarded in particular later during the evaluation of the classifiers.

To show that the hypotheses stated above, namely that ***syngrouping*** and ***compgrouping*** enhance the efficiency of the baseline method, are correct, the following two tables, Tables 7 and 8 , show the summed up results.

	Base1	Base2	Base3
Results with only ***syngrouping***	39/117 (34%)	59/117 (50%)	62/117 (53%)
Results with only ***compgrouping***	39/117 (34%)	52/117 (44%)	61/117 (52%)

Table 7: Reduced baseline precision with one grouping type, strict annotation

	Base1	Base2	Base3
Results with only ***syngrouping***	43/117 (37%)	65/117 (56%)	69/117 (59%)
Results with only ***compgrouping***	45/117 (38%)	59/117 (50%)	70/117 (60%)

Table 8: Reduced baseline precision with one grouping type, liberal annotation

Each of the groupings shows an increase of overall performance by several percent, thus supporting the discussed hypotheses about the grouping methods increasing the performance.

The ***compgrouping*** heuristic has also been validated by reviewing the created groups. Three annotators had to identify the largest subset of terms that had to be connected by any two words forming a pair, in which one term was a composition starting or ending with the other term with its true word sense. For example "Restaurant" and "Hotel-Restaurant" would be such a pair, because the latter end with the former. Likewise "Hotel-Restaurant" and "Hotel" would be a valid pair, since "Hotel-Restaurant" begins with "Hotel". All 3 terms would belong to the same group because there is a connection from "Restaurant" with "Hotel" via the term "Hotel-Restaurant". An unwanted pair would be for example "Buch" and "Bucht", though the latter starts with the former fulfilling the requirements for the grouping, the two terms have nothing in common, since the "Buch"-part of "Bucht" does not have the same meaning as "Buch" on its own.

The review of the formed groups returned 90.4%, 93.4% and 95.8% related words in such groups, averaging the error to 6.8% (this has been measured for grouping performed on all of the 117 named entities this evaluation is based on).

Improving the baseline by including both grouping strategies, and thereby creating the final baseline algorithm to compare against, yields the results presented in Tables 9 and 10.

	Base1		Base2		Base3	
	Training set	Test set	Training set	Test set	Training set	Test set
Localities	18/40 (45%)	14/30 (47%)	21/40 (53%)	18/30 (60%)	26/40 (65%)	19/30 (63%)
Misc. domains	6/27 (22%)	5/20 (25%)	9/27 (33%)	7/20 (35%)	11/27 (41%)	8/20 (40%)

Sum	43/117 (37%)	55/117 (47%)	64/117 (55%)

Table 9: Full baseline precision, strict annotation

For the strict annotation an enhancement of 7% for Base1, 0% for Base2 and 5% for Base3 has been achieved by the applied grouping.

	Base1		Base2		Base3	
	Training set	Test set	Training set	Test set	Training set	Test set
Localities	20/40 (50%)	14/30 (47%)	22/40 (55%)	19/30 (63%)	28/40 (70%)	21/30 (70%)
Misc. domains	8/27 (30%)	7/20 (35%)	12/27 (44%)	9/20 (45%)	14/27 (52%)	10/20 (50%)

Sum	49/117 (42%)	62/117 (53%)	73/117 (62%)

Table 10: Full baseline precision, liberal annotation

Grouping leads to a total increase of overall precision of 9% for Base1, 1% for Base2 and 6% for Base3 with the liberal annotation.

A special aspect concerning the strict annotation is that for some of the 117 named entities not a single correct answer had been extracted from the corpus. This is the case for 3 named entities from the training set (1 locality and 2 miscellaneous) and 1 named entity from the test set (also a locality), thus triggering a guaranteed failure for the testing of the corresponding named entity.

Since the classification-performance using ML-methods is only meaningful, when examined on the test set (the classifiers are calibrated using the training set and therefore are supposed to work well on it). A reasonable comparison to the baseline therefore has to be compared with the baseline's performance on the same set. These values are presented in the following Table 11.

	Base1	Base2	Base3
Test set (strict)	19/50 (38%)	25/50 (50%)	27/50 (54%)
Test set (liberal)	21/50 (42%)	28/50 (56%)	31/50 (62%)

Table 11: Baseline performance on the training set

3.4 Classifiers' performance

In this section the precision of different classifiers will be evaluated. It starts with analyzing the parts of the input vector that are non-structural tags (structural tags are POS-tags and chunk tags and ***WordDistance***, as described in Section 2.4.2). Those non-structural features are the ***Hearst-patterns***, hints on the occurrence of an anaphora (***anaphora-hints***) and whether a hypernym candidate or the named entity itself appears in the document title (***Hypernym-in-Title***, ***NE-in-Title***). A qualitative estimation of the influence of each feature will be given followed by an experiment to verify the estimation. I will continue with the evaluation of the performance of the classifiers with varying snippet sizes and finish the evaluation by presenting the best results reached by the classifiers that have been assessed.

3.4.1 Non structural features

In addition to the classification parameter, four non structural features have been introduced in Section 2.4.2:

- A feature that indicates a hypernym candidate being part of a Hearst-pattern in a snippet.
- One that possibly indicates the use of an anaphora.
- Two features to specify whether the named entity is part of the titles or whether the hypernym candidate is part of the titles

To get a picture on the probability of each feature to hold the value true, the probabilities for each feature will be presented after the definition of a few terms. The following boolean variables will be used:

C	:=	true if the ***Classification*** feature of a training vector is set to “true”
H	:=	***Hearst-pattern*** feature of a training vector is set to “true”
A	:=	***Anaphora-hint*** feature of a training vector is set to “true”
TH	:=	***Hypernym-in-Title*** feature of a training vector is set to “true”
TE	:=	***NE-in-Title*** feature of a training vector is set to “true”

P(C) is defined as the probability of the Classification feature of an input vector from the set of training vectors (training vectors are derived from the snippets belonging to the training set of named entities) being set as true. The other probabilities P(H) etc. are defined likewise. With these probabilities defined and calculated for the test set of vectors, we get the following results:

Number of vectors: n = 11'754

Feature (X)	***P(X) * n***	***P(X) /(%)***	***P(C ∣ X) /(%)***
C	1044	8,88	100
H	75	0,64	44,0
A	76	0,65	5,3
TH	2419	20,6	21,3
TE	9521	81,0	8,8

Table 12: Distribution of non-structural features, strict annotation, training set

The combination of two parameters are relevant for a feature X to have an influence on the classification. The first one is how frequently a feature can be observed in the samples and the other one is how strongly it biases a decision, i.e. a feature that appears seldom (P(X) small) will only have a small influence on the average and likewise, if it appears quite frequently, but has the same distribution like all positive examples (P(C ∣ X) ≅ P(C)), it should not have a strong impact on the classification either.

Keeping this in mind when looking at the data, it should be possible to make a more or less good estimate towards the importance of a feature. In Table 12 we notice 2 features with a very low frequency, these are the Hearst-patterns and the Anaphora-hints. Their frequencies are even below 1%, thus I would conclude a rather low impact on classification decisions. Though the Hearst-Patterns, in contrast to the Anaphora-hints, bias a vector strongly to be correct 44% for P(C ∣ H) compared to an average of about 9% correct samples.

The other two vectors occur rather frequently. While P(C ∣ TE) does not differ strongly from the average distribution of correct vectors P(C), I would estimate a perceivable difference, when adding the NE-in-Title feature.

The same estimations hold for the results of tests under conditions of liberal annotation. For completeness following the corresponding figures:

Feature (X)	*P(X) * n*	*P(X) /(%)*	*P(C ǀ X) /(%)*
C	1377	11,7	100
H	75	0,64	45,0
A	76	0,65	9,2
TH	2419	20,6	25,0
TE	9521	81,0	11,7

Table 13: Distribution of non-structural features, liberal annotation, training set

However, these estimations do not count in possible correlation with the structural features, so results may very well show results, that do not reflect the estimations made. Additionally, the way the machine learning methods will combine features, will surely have an influence on the resulting classification. The selection function that will be used, adds yet another parameter that will influence the performance.

To verify the estimations above the results of the classification, only using the structural (POS-tags, chunk tags and WordDistance) features will be the base to compare against (See Figure 5 and 6 for the basic results). Following that base feature set, each time will be amended by a single feature and results will be compared.

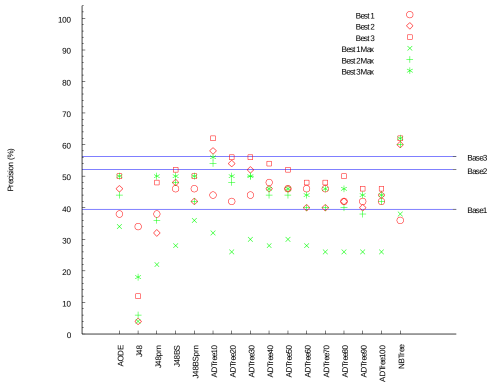

Figure 5: Classification using only structural features, strict annotation

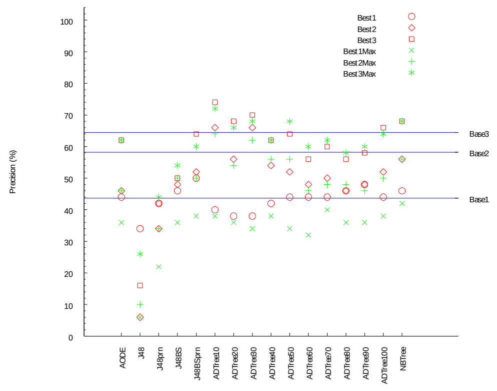

Figure 6: Classification using only structural features, liberal annotation

The x-axis shows the different classifiers that have been used. Those are:

- AODE, a Bayesian Networks based method.
- J48, a Decision Tree based method. There are 4 versions of it because 2 parameters can be set changed. A “BS” in the name means binary splitting has been activated and “prn” in the name indicates that pruning has been enabled.
- ADTree, also a Decision Tree based algorithm, has been tested with various numbers of boosting iteration. The number behind the name denotes the iteration count.
- NBTree is a method combining Decision Trees and Bayesian Networks.
- SMO is a Support Vector Machine implementation.

The SMO method has been evaluated only in a few tests to compare it with the other classifiers, because it is very computing intensive and in the tests it was applied in, it did not perform very well. The reasons for this has partly been discussed in the introduction of the target function (Section 2.4.2).

Some strange artifacts can be observed in a couple of graphs: Sometimes the value for BestN is higher than the one for BestN+1, whereas one would reasonably anticipate monotonically increasing values for increasing N. This effect is a result of the construction of the selection functions, which treats equally ranked hypernym candidates by choosing the first one. This choice has been made because as such the system will produce the same results with every run, while a random choice would introduce some form of nondeterminism making the comparison of results more difficult.

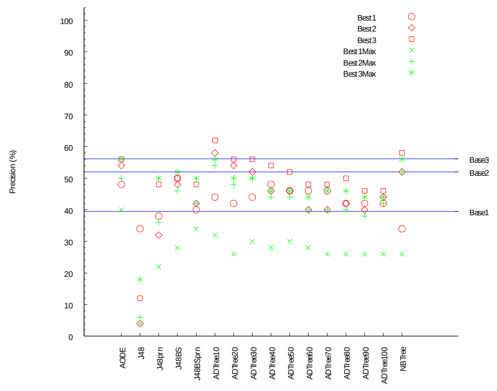

Figure 7: Classification using structural and Hearst-pattern features, strict annotation

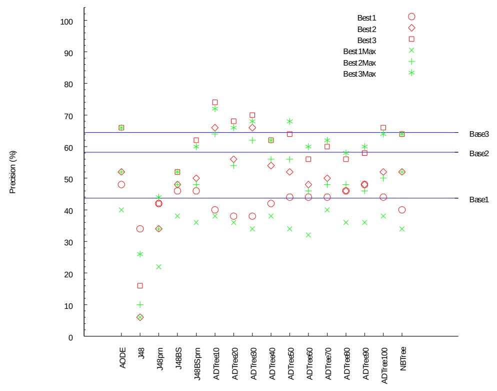

Figure 8: Classification using structural and Hearst-pattern features, liberal annotation

The results, when including the Hearst-pattern feature, show an average increase of Best1Max' performance, thus indicating that the direct classification performance has increased. That means, rates assigned to some occurrences of hypernym candidates became better, while the average assignment for which Best1 is an indicator, seems to have decreased. Especially J48 has gained by the addition of the feature, while the good performance of ADTree10 and NBTree dropped. And maximum performance dropped slightly.

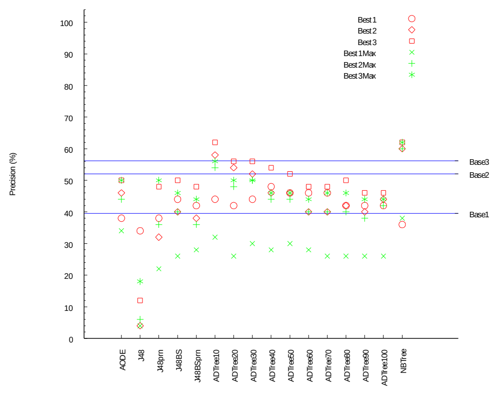

Figure 9: Classification using structural and anaphora-hint features, strict annotation

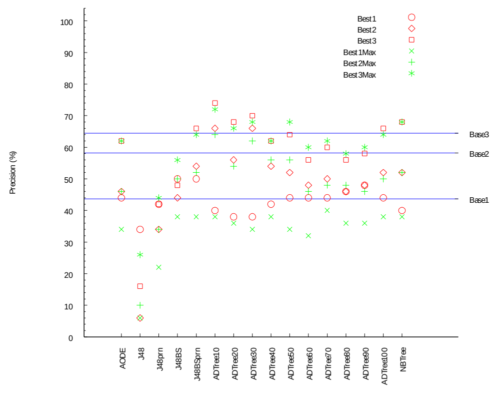

Figure 10: Classification using structural and anaphora-hint features, liberal annotation

The results reached with the inclusion of the Anaphora-hint feature show nearly the same results. J48 lost slightly in performance with strict annotations, but gained a bit with liberal annotations. With liberal annotations the Bayesian based methods, AODE and NBTree lost up to 6%. On average, though, the results did not change considerably, which is in accordance with the estimation.

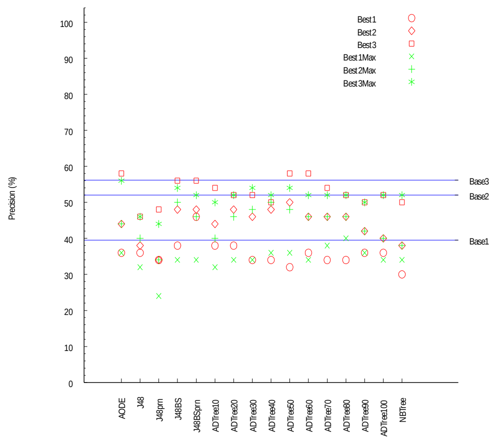

Figure 11: Classification using structural and Hypernym-in-Title features, strict annotation

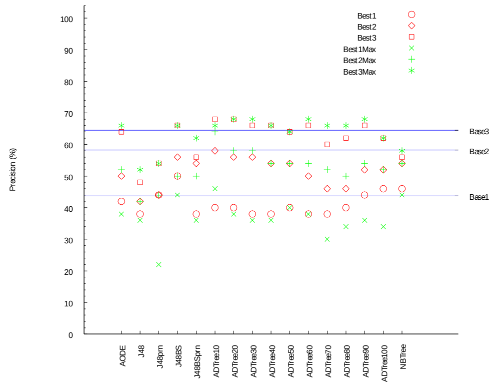

Figure 12: Classification using structural and Hypernym-in-Title features, liberal annotation

Some effect was anticipated with the addition of the Hypernym-in-Title feature. Analyzing the graphs shows a consequence that in fact, the results differ to a considerable extent. Remarkable is that precision for BestN mostly decreased, while the direct selection of classification results (BestNMax) resulted in enhancements of sometimes more than 20% using the liberal annotation. J48 performed much better, AODE gained a bit and NBTree lost some precision, while for the ADTrees just the optimal number of boosting iterations changed.

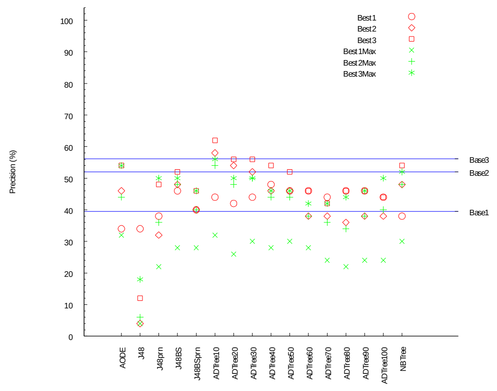

Figure 13: Classification using structural and NE-in-Title pattern features, strict annotation

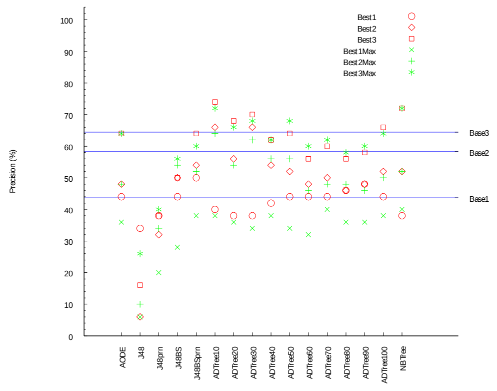

Figure 14: Classification using structural and NE-in-Title pattern features, liberal annotation

The last feature to be assessed is the NE-in-Title feature. This one with its distribution being nearly the same as the one of positive to negative examples and occurring very frequently (about 80%) could indicate the loss in performance, when adding a feature that doesn't introduce any new information. With strict annotations chosen, an average decrease of about 5% can be perceived for those classifiers affected. ADTrees seem to be rather robust and are only minimally affected. With liberal annotation the effect was less, the Bayesian-based AODE even showed better values. As estimated the feature did convey a gain in performance, rather a decrease by increasing dimensionality.

Below in Table the maximum values of any classifier are listed for easier comparison.

	Best1	Best1 Max	Best2	Best2 Max	Best3	Best3 Max
Baseline	19 (38%)		25 (50%)		27 (54%)	
Classifier (structural)	24 (48%)	19 (38%)	30 (60%)	30 (60%)	31 (62%)	31 (62%)
Classifier (+ Hearst-pattern)	25 (50%)	20 (40%)	30 (60%)	30 (60%)	31 (62%)	31 (62%)
Classifier (+ Anaphora-hint)	24 (48%)	17 (34%)	29 (58%)	27 (54%)	31 (62%)	28 (56%)
Classifier (+ Hyp-in-Title)	23 (46%)	19 (38%)	25 (50%)	25 (50%)	29 (58%)	28 (56%)
Classifier (+ NE-in-Title)	24 (48%)	16 (32%)	29 (58%)	27 (54%)	31 (62%)	28 (56%)

Table 14: Feature comparison, strict annotation

	Best1	Best1 Max	Best2	Best2 Max	Best3	Best3 Max
Baseline	19 (38%)		25 (50%)		27 (54%)	
Classifier (structural)	25 (50%)	20 (40%)	33 (66%)	32 (64%)	37 (74%)	36 (72%)
Classifier (+ Hearst-pattern)	24 (48%)	20 (40%)	33 (66%)	32 (64%)	37 (74%)	36 (72%)
Classifier (+ Anaphora-hint)	25 (50%)	20 (40%)	33 (66%)	32 (64%)	37 (74%)	36 (72%)
Classifier (+ Hyp-in-Title)	25 (50%)	23 (46%)	29 (58%)	32 (64%)	34 (68%)	34 (68%)
Classifier (+ NE-in-Title)	25 (50%)	20 (40%)	33 (66%)	32 (64%)	37 (74%)	36 (72%)

Table 15: Feature comparison, liberal annotation

The estimation, it seems, could not exactly be verified. The Hearst-patterns changed the results slightly to the better, as thought. But all other features resulted in a noticeable decrease in precision (especially when only looking at the best results). To better understand this result, we will have to look at the distribution of patterns in the test set.

Number of vectors: n = 8084

Feature (X)	***P(X) * n***	***P(X) /(%)***	***P(C | X) /(%)***
C	866	10,7	100
H	71	0,88	62,0
A	60	0,74	5,0
TH	1283	15,9	46,6
TE	5118	63,3	11,6

Table 16: Distribution of non-structural features, strict annotation, test set

Feature (X)	***P(X) * n***	***P(X) /(%)***	***P(C | X) /(%)***
C	1046	12,9	100
H	71	0,88	66,2
A	60	0,74	5,0
TH	1283	15,9	57,0
TE	5118	63,3	14,3

Table 17: Distribution of non-structural features, liberal annotation, test set

If we compare the distribution of features from the test vectors in Tables 16 and 14 just above, with those extracted from the training vectors, similar aspects can be noticed as in the training vectors, just with a higher deviation. A hypernym candidate appearing in a title or a Hearst-pattern being found makes the hypernym candidate even more probable to be a correct hypernym than can be concluded from the data of the training set. The other two features do not display any remarkable differences. Hence I would conclude, that the addition of an additional dimension is responsible for the biggest part of decrease in performance.

Support Vector Machines being quite computing intensive, they now will be regarded detached from the preceding analysis. For comparison the results from tests with a snippet length of eleven tokens will be regarded. The following two Figures, 15 and 16 show performance with all features enabled.

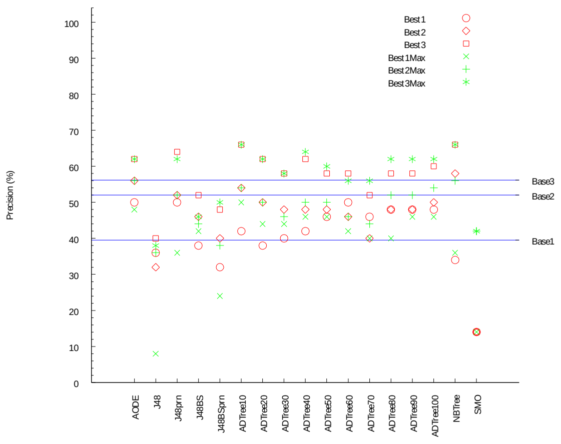

Figure 15: SVM test with SnippetRadius 5, strict annotation

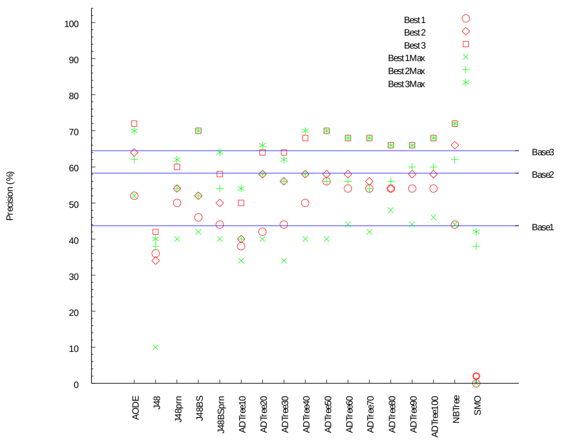

Figure 16: SVM test with SnippetRadius 5, liberal annotation

A glance at the graphs shows a very low performance for the SVM. This can result from multiple reasons. First of all tuning the parameters of the SVM might increase precision. Though I tend to follow another direction of argumentation. SVMs classifiers normally map to the domain {true, false}, so there is no value between zero and one, no uncertainty. But, as has been discussed, many constellations in natural language can lead to multiple interpretations. Thus some uncertainty should add to build an adequate model.

3.4.2 Snippet sizes

One would tend to think that "the more information is added to a learning system, the better the result will be, hence the longer a snippet the better the result". This is not necessarily true because of at least two important arguments. The first point would be, that it is possible that words describing the named entity are normally very close to it, thus possibly rendering the additional information useless. Secondly the number of samples that have to be presented to a classifier to get feasible results, increases with the dimensionality of the input vector and it's components. Thus different snippet sizes will be evaluated in this section.

While the above evaluations have been performed using snippets consisting of 21 tokens, the performance considering smaller snippets will be shown below. Looking at larger snippets here is not an option, because new nouns would be extracted that have not been manually classified and therefore testing them would result in a certain failure.

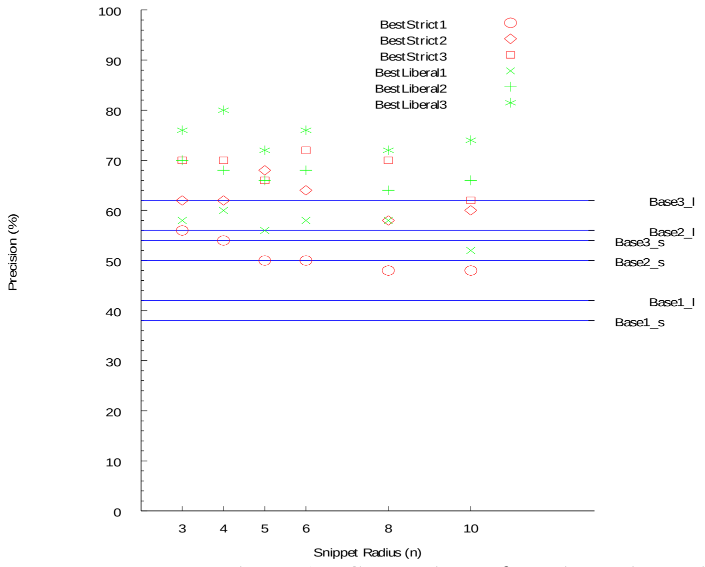

Figure 17: Comparison of varying snippet sizes

Figure 17 shows the maximum results (maxed over all classifiers' results) for different snippet sizes. The ***SnippetRadius*** represents the number of words to either side of the named entity, that have been part of a snippet and the functions ***BestStrictN*** and ***BestLiberalN*** constitute the maximum results reached with one of the corresponding ***BestN*** or ***BestNMax*** function in the given setup.
Opposed to the above stated intuitive thought, that more information results in better precision, has clearly been falsified, since smaller snippet sizes in this setting return

higher precisions.

Speaking of high precisions now, leads us directly to the final comparison of results with the performance of the baseline method.

	Best1(Max)	Best2(Max)	Best3(Max)
Test set (strict)	28/50 (56%)	32/50 (64%)	36/50 (72%)
Overall increase	9/50 (18%)	7/50 (14%)	9/50 (18%)
Test set (liberal)	30/50 (60%)	35/50 (70%)	40/50 (80%)
Overall increase	9/50 (18%)	7/50 (14%)	9/50 (18%)

Table 18: Maximum classifier performance

4 Discussion

The results discussed above state a clear picture. The goal to demonstrate, that machine learning is a viable approach to identify hypernyms for named entities from the immediate surrounding of the latter, has been achieved. It could be shown, that the application of ML-based methods to this problem surpasses the baseline method by 18% using a set of strictly correct hypernyms that had to be extracted in order to result in a successful classification, resulting in an increase from 38% to 56% precision. Additionally allowing the extraction of plural forms and the four cases of the German language, likewise an increase in performance by an absolute 18% could be achieved, augmenting the 42% precision reached by the baseline method to a precision of 60%.

Furthermore, evidence suggests that the amount of surrounding information taken into account, is an important factor influencing the precision of hypernym extraction.

And since the corpus consisted of a set of web pages retrieved by means of the search engine Google^{TM}, it could also be shown, that the use of web resources in combination with a potent search engine delivers promising results.

4.1 Error and error-propagation

Many of the different methods applied at the various stages of data processing introduce errors. This starts at the first steps of preprocessing. The extraction of natural language from HTML is the initial step that heavily contributes to the overall error, making the decision of what is actually part of natural language and what is not. After this step artifacts reside in the text that reduce the performance of the POS-tagger and thus results in a loss in accuracy of the chunker. Regrettably the exact errors are not known and especially the correlations between them are unknown such that a quantitative error calculation can not be presented. Though I would suppose that the earlier an error is introduced, the heavier its influence will be. As such it might be very interesting to apply this system to a set of documents that do not have to be extracted from HTML, possibly even being manually tagged with POS-tags and chunk tags. That might be an approach to get a clearer picture of error propagation.

4.2 Further research

To enhance the performance of the presented system, in addition to reducing the errors just discussed, a number of enhancements could be added.

Further text enrichment may be achieved by the employment of a stemmer, which would assign word stems to each word. This might help to find more synonymy relations, and thus, by adding up their ratings as done by the BestN selection function, better results might be achieved. Moreover, additional thought could be spent on the design of selection functions (discussed in Section 2.6), because when looking at the evaluation, considerably differing results, depending on the applied function, can be observed.

A named entity taggers, offering precisions in the high 90%, would probably increase the performance of filtering words that most probably are not hypernyms. Since normally named entities are not classes of objects, they cannot be hypernyms. Another way to filter out words that are named entities are gazetteers, lists of words like street names, country names or town names. Heavy use of those might also increase performance.

A totally different approach would be to identify classes of verbs that indicate a likely class of hypernyms. Likewise closed tag sets, i.e. tag sets with only a small number of words, like determiners, could be further exploited. An example might be “Peter and Betty are having dinner at the Golden Dragon”. From this sentence, we might infer, that “the Golden Dragon” is a place where people go for having dinner, probably a restaurant. This is not a necessary conclusion, but a likely one, after all “the Golden Dragon” might be an animal or a park. Though if many such examples indicate, that “the Golden Dragon” is a restaurant, it might as well be true.

5 Bibliography

(Adams): D. Adams, The Hitchhiker's Guide to the Galaxy, Pan Books, 1979

(Brants): T. Brants, W. Skut, H. Uszkoreit, Syntactic annotation of a German newspaper corpus, In Proceedings of the ATALA Treebank Workshop, 1999

(Cederberg): S. Cederberg, D. Widdows, Using LSA and Noun Coordination Information to Improve the Precision and Recall of Automatic Hypernymy Extraction, Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL, 2003

(Cimiano): P. Cimiano, S. Staab, Learning by Googling, SIGKDD Explorations, 2004

(Evans): R. Evans, A Framework for Named Entity Recognition in the Open Domain, Proceedings of Recent Advances in Natural Language Processing, Borovetz, Bulgaria, 2003

(Feldbaum): C. Feldbaum, An Electronic LexicalDatabase, MIT Press, 1989

(Florian): R. Florian, A. Ittycheriah, H. Jing, T. Zhang, Named Entity Recognition through Classifier Combination, Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL, 2003

(Freund): Freund, Y., Mason, L., The alternating decision tree learning algorithm, Proceeding of the Sixteenth International Conference on Machine Learning, Bled, Slovenia, 1999

(Hearst): M.A. Hearst, Automatic acquisition of hyponyms from large text corpora, Proceedings of the Fourteenth International Conference on ComputationalLinguistics, Nantes France, 1992

(Kilgarriff): A. Kilgarriff, Web as corpus, Proceedings of Corpus Linguistics, 2001

(Kohavi): R. Kohavi, Scaling up the accuracy of naive-Bayes classifiers: a decision tree hybrid, Proceedings of the Second International Conference on Knowledge Discovery and Data Mining, 1996

(Manning Schütze): C. D. Manning, H. Schütze, Foundations of Statistical Natural Language Processing, MIT, 1999

(Mitchell): T.M. Mitchell, Machine Learning, McGraw-Hill Companies, Inc., 1997

(Platt): J. Platt, Fast Training of Support Vector Machines using Sequential Minimal Optimization, MIT Press, 1998

(Porzel Malaka): R. Porzel, R. Malaka, Towards Measuring Scalability in Natural Language Understanding Tasks,Proceedings of the HLT-NAACL, Boston, Massachusetts, USA, 2004

(Quinlan): R. Quinlan, C4.5: Programs for Machine Learning, Morgan Kaufmann Publishers, San Mateo, CA, 1993

(Shannon): C. E. Shannon, A Mathematical Theory of Communication, The Bell System Technical Journal, 1948

(van Rijsbergen): C.J. van Rijsbergen, Information Retrieval, University of Glasgow, 1979

(Webb): G. Webb, J. Boughton, Z. Wang, Averaged One-Dependence Estimators: Preliminary Results, AI2002 Data Mining Workshop, Canberra, 2002

6 Appendix

6.1 List of Figures

6.2 List of Tables

6.3 List of named entities

Below the named entities used for the evaluation will be listed. Each entry consists of the named entity itself separated by “//” from a contextual term that has been used in the Google™ query as context information during the text acquisition.

6.3.1 Locality domain

Training set:

```
Alt Hendesse // Heidelberg
Vetter// Heidelberg
Billy Blues// Heidelberg
Kleiner Mohr// Heidelberg
Médoc// Heidelberg
Roter Ochsen// Heidelberg
Thanner// Heidelberg
Vater Rhein// Heidelberg
La Maison// Heidelberg
Havanna// Heidelberg
Mesón Madrid// Heidelberg
Olive// Heidelberg
Weißes Rössel// Heidelberg
Indian Palace// Heidelberg
Kupferkanne// Heidelberg
Backmulde// Heidelberg
Palmbräugasse// Heidelberg
Zum Seppl// Heidelberg
La vie en rose// Heidelberg
Dimitra// Heidelberg
Hellas// Heidelberg
Zur Krone// Heidelberg
Scala// Heidelberg
Destille// Heidelberg
Erbprinzen// Heidelberg
Zur Scheune// Heidelberg
Hard-Rock// Heidelberg
Le Coq// Heidelberg
Molkenkur// Heidelberg
Subway// Heidelberg
Taormina// Heidelberg
Zum Schwarzen Walfisch// Heidelberg
Burkardt// Heidelberg
Boulevard// Heidelberg
Merlin// Heidelberg
```

```
Kaschmir// Heidelberg
Print Media Lounge// Heidelberg
Regie// Heidelberg
Zimmermann// Heidelberg
Zum Binsebub// Heidelberg
```

Testset:

```
Hörnchen// Heidelberg
Zur Züchterklause// Heidelberg
Olympia// Heidelberg
Sayang// Heidelberg
Zum Mohren// Heidelberg
Zum Weissen Schwanen// Heidelberg
Karlsberg Pick// Heidelberg
Zum Spreisel// Heidelberg
Santa Lucia// Heidelberg
Bergstraße// Heidelberg
Schwarzer Peter// Heidelberg
Krokodil// Heidelberg
Auerstein// Heidelberg
Schwalbennest// Heidelberg
Adlerstübchen// Heidelberg
Lavazza// Heidelberg
Simplicissimus// Heidelberg
Schultes// Heidelberg
Ellin// Heidelberg
Adler// Heidelberg
Belcanto// Heidelberg
Akademie// Heidelberg
La Vite// Heidelberg
Piccolo Mondo// Heidelberg
Schnitzelbank// Heidelberg
Hutzelwald// Heidelberg
Brasserie Fritz// Heidelberg
Orchid Royal// Heidelberg
Sakura// Heidelberg
Golden Nugget// Heidelberg
```

6.3.2 Miscellaneous domains

Training set:

```
Weihnachten  //
Tibet //Geographie
Maori  //Ethnologie
Ferrari  //Autorennen
Challenger   //Raumfahrt
Mahatma Ghandi   //Indien
Flipper    //Tiere
Karl der Große  //Mittelalter
Allianz   // Geld
Henkel  //Waschmittel
Mars //Lebensmittel
Bounty  //Seefahrt
Wasser  //Leben
Straßburg  //Frankreich
Cannondale  //Radsport
Kriemhild  //Literatur
Springer  //Medien
King Kong //Unterhaltung
Tiefseetaucher   //Unterhaltung
Michael Schumacher  //Boulevard
Der Schimmelreiter   //Literatur
Meningitis // Medizin
blaue Mauritius // Post
Forelle blau // Küche
Cordon Bleu // Küche
Hundertwasser  //Kunst
Faserland  //Literatur
```

Test set:

```
Rübezahl   //Mythologie
Ascorbinsäure // Chemie
Alraune  //Natur
Michelin  // Motorsport
Testosteron   //Chemie
Bianchi // Radsport
Bloody Mary  //Geschichte
Gorch Fock   //Meer
Monte Alban //Geschichte
Calzone // Küche
Alpha Centauri   //Astronomie
Annette von Droste-Hülshoff  // Literatur
```

```
Mount Kailash  //Geographie
Chemie  //Lehre
Nike //Mythologie
Rußpartikel   //Umwelt
Twix  //Lebensmittel
Glycin // Biologie
Günther Strack  //Literatur
Hepatitis // Medizin
```

6.4 German Hearst-patterns

Extract from the source code:

```
/* Hearst Patterns for German:
 *
 * Definitions:
 * <X>            ::= Suchwort (named entity)
 * <X?>           ::= Suchwort (named entity) or other named entity
 * <Y>            ::= Lösungswort (hypernym) (in upper case!!!)
 * <BEST_ART>     ::= {[Dd]er | [Dd]ie | [Dd]as}
 * <UN_BEST_ART>  ::= {[Ee]in | [Ee]ine}
 * <BEST_ART_G>   ::= {[Dd]es | [Dd]er}
 * <KON>          ::= {und | oder}
 * <SPEZIAL1>     ::= {wie {z.B.}? | insbesondere | besonders}
 * <SPEZIAL2>     ::= {einschließlich}
 * <SPEZIAL3>     ::= {andere | ähnliche | weitere}
 *
 * Patterns:
 * <BEST_ART> <X> <Y>
 * <BEST_ART> <Y> <X>
 * <Y>, <SPEZIAL1> {<BEST_ART> <X?>, }* <BEST_ART> <X?> {<KON>
<BEST_ART> <X?>}?
 * <Y>, <SPEZIAL2> {<BEST_ART_G> <X?>, }* <BEST_ART_G> <X?> {<KON>
<BEST_ART_G> <X?>}?
 * <X> <KON> <SPEZIAL3> <Y>
 * <X>, {<BEST_ART> | <UNBEST_ART>} <Y>
 * <X> ist <UNBEST_ART> <Y>
 *
 */
```

6.5 About the platform

Nearly all of the program developed is written in Java and uses Java based external Libraries. One exception exists, the POS-Tagger TreeTagger, which is written in C and can be downloaded as an executable from the University of Stuttgart. At the heart of the machine learning part WEKA, a framework for machine learning and data mining created mainly at the University of Waikato, New Zealand, is the system of choice.

Where[25] to get the sources / binaries:

WEKA

http://www.cs.waikato.ac.nz/ml/weka/

TreeTagger

http://www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger/DecisionTreeTagger-de.html

Qtag (had been integrated but has been neglected in favor of TreeTagger)

http://www.english.bham.ac.uk/staff/omason/software/qtag.html

OpenThesaurus

http://www.openthesaurus.de/

Google™-API

http://www.google.com/apis/

25 All addresses last accessed in August 2005

6.6 Alphabetical Index