Entity-based information retrieval

Andreas Salomonsson

May 14, 2012

Abstract

The goal of this master's thesis is to extract structured semantic information from Swedish text documents. This information is then intended to be stored as metadata alongside the original text document. Making this metadata searchable should then create a more powerful search engine, both in the sense that it allows for more complex queries and it can be used to give more relevant results.

To achieve this, we explored different methods of carrying out named entity recognition. The named entities found in a text can then be used to extract structured information from semantic networks. We associated a unique identifier in a semantic network with each found named entity.

The result of this thesis is a program that takes plain text as input and outputs structured information about the entities in the text. We have evaluated the performance of the different parts of the program and compared it to existing systems.

Acknowledgements

I would like to thank my examiner, Pierre Nugues, and my supervisor, Svetoslav Marinov, for their support any many interesting discussions and suggestions.

I would also like to thank everyone at Findwise's office in Copenhagen; you have all been very supportive.

Contents

Chapter 1

Introduction

In this chapter, we give an introduction to the extraction of entities from texts, including a background, a goal, and previous work. We also present the implementation framework used.

Searching for documents and finding relevant documents is a common task. As the amount of documents increase finding the most relevant documents becomes harder. One of the most common ways of searching for documents is by creating an index for all documents that is to be searchable. A user can then use text strings to query the index in order to find documents containing the given words. A user of such a search engine has to then think of what words exist in the documents the user is interested in. After a first query attempt, the user might find out that most of the top ranked search results have a completely different topic than the one that is wanted. The reason for this is, of course, that the same word can have more than one meaning. In a second attempt, the user might try to add or remove search terms as to avoid the unwanted documents. After a few attempts, the user would probably get decent results, but there is no way of knowing if interesting documents have been missed that just do not contain the words searched for.

One way to address this shortcoming is to try and capture the semantics of the words. Let us say one is interested in finding documents about the current American president. Obvious search terms include 'president', 'Barack', and 'Obama'. Those search terms are likely to give rather good results, but further down in the result list it is likely that there will be documents about other presidents for example. It would be possible to require all the documents to contain all the search terms to avoid other presidents as results. Such a query would miss documents containing, for example, 'the current president', 'Barack Obama', or even just 'Obama'. If such entities, Barack Obama in this case, were to be identified at index time they could be mapped to a uniquely identifiable entity. If there were other information available about that entity, for example that he is the current American president; it would be possible to avoid documents not containing the current president. This is just one example of what semantic information about named entities could be used for. A more complex example could be to search for all the restaurants within two kilometers, from a given location, that serves Thai food. This is something that is really hard to find by just looking at text strings.

1.1 Background

In classical textual search, the goal is to find a given text string in a document or all documents containing a certain search term. This type of search has some limitations. Often when a person searches for something, the semantics is much more important to the user compared to the text strings themselves. One of the reasons for this is that natural language is in many ways ambiguous. For example, it is very common for a word to have more than one meaning. In this work we propose capturing of the semantics in texts and search queries in order to give better results.

One step in trying to extract more information from a text that is very common is to tag each word with its corresponding part of speech. This alone can be used, to a certain degree, to disambiguate the meaning of a word.

Another important aspect is to extract named entities in texts, for example names of persons, locations, organizations, and so on.

Entities and relations between them stored in a semantic network can be very useful after a mapping has been done from extracted entities in a text to a unique identifier in a semantic network.

1.2 Aim of this thesis

The aim of this thesis is to extract named entities in documents written in Swedish and extract information about them from semantic networks. The text documents we have focused on are Swedish news articles and medical texts. When the entities have been disambiguated and mapped to unique identifiers in a semantic network, a lot of other information is made available. This other information can then be stored alongside the original text document as metadata. Making the metadata searchable would result in a more powerful search compared to a textual search.

1.3 Previous work

Accurate part-of-speech taggers exist for most of the natural languages used today. Since the focus of this thesis is on Swedish texts there are at least three options available:

- Granska Tagger: A Part-of-Speech Tagger for Swedish (Carlberger and Kann, 1999) is a part-of-speech tagger developed at the Royal Institute of Technology in Stockholm.
- HunPos is a reimplementation of Tnt (Brants, 2000) and it is described by Halácsy et al. (2007).
- The part-of-speech tagger available in OpenNLP[1], an Apache project concerned with using machine learning techniques to process natural language.

There are a number of available Named Entity Recognition (NER) systems available. For example:

[1] http://http://opennlp.apache.org/

- Stanford Named Entity Recognizer (Finkel et al., 2005)
- Illinois Named Entity Tagger (Ratinov and Roth, 2009)

Unfortunately none of those can directly be used to process Swedish texts. Language independent NER systems were also developed in the shared task of CoNLL 2002[2] and 2003[3]. The performances of all the competing systems are also available.

There are a few projects that have been concerned with storing structured information in semantic networks. DBPedia[4] is a large semantic network where information has been extracted from Wikipedia articles. Another semantic network is YAGO2 (Hoffart et al., 2011a), which in some ways overlaps DBPedia since the information found in YAGO2 comes from Wikipedia[5], Geonames[6], and WordNet[7]. In Geonames, a large set of geographical locations are available alongside with alternative names and coordinates. WordNet is a project where all meanings for all nouns, verbs, adjectives, and adverbs in English have been given a unique semantic identifier. Both DBPedia and YAGO2 are language independent in the sense that all entities have a unique identifier that does not depend on the language. This makes it useful for Swedish texts, as well as for other languages than English, since some entries in the semantic networks are language specific. There is, for example, a set of alternative names in different languages related to the unique identifier for an entity.

Named entity disambiguation is the task of finding a unique identifier for an entity that has been found. Previous work concerned with this task includes Hoffart et al. (2011b).

To apply machine learning techniques, training data needs to be available. Swedish training data is available in the Stockholm-Umeå Corpus (SUC) version 2.0, which is a balanced corpus, annotated with part-of-speech tags, morphological features, and named entities among other things. Thus, SUC can be used to train both part-of-speech taggers and named entity recognition systems. The details of the corpus can be found in the manual (Gustafson-Capková and Hartmann, 2006).

1.4 Implementation framework

We wrote all software in Java and external programs and libraries have also been written in Java, with the only exception being HunPos. The input text needs to be processed in a number of different steps. Thus, we chose to use a pipeline architecture. The first thing we did was to try and get every step working as fast as possible. When we then had a working system available, improvements were made to the individual steps.

We have developed a graphical user interface to help visualize the results of the thesis. In this interface, the input and output from each step can easily be examined, which has been really useful in giving hints about the performance of the system.

[2] http://www.cnts.ua.ac.be/conll2002/ner/
[3] http://www.cnts.ua.ac.be/conll2003/ner/
[4] http://wiki.dbpedia.org/About
[5] http://www.wikipedia.org/
[6] http://www.geonames.org/
[7] http://wordnet.princeton.edu/

We first implemented a tokenizer and sentence detector, since the input to HunPos should be one token per line and sentences separated with empty lines. Now with part-of-speech tags and morphological features for each token, we developed a rather simple NER step based on hand-written rules. In the NER step we added a tag for each token, showing whether or not it is part of a named entity. We then mapped the found entities to unique identifiers in the semantic network and finally other information was extracted about the found entities. To disambiguate in the case where more than one unique identifier was found for an entity, we used the simple approach of choosing the one with the most information available. Figure 1.1 shows an overview of the pipeline architecture. When we had a working system, we improved some parts of the first version of it, according to what is stated in Chapter 4.

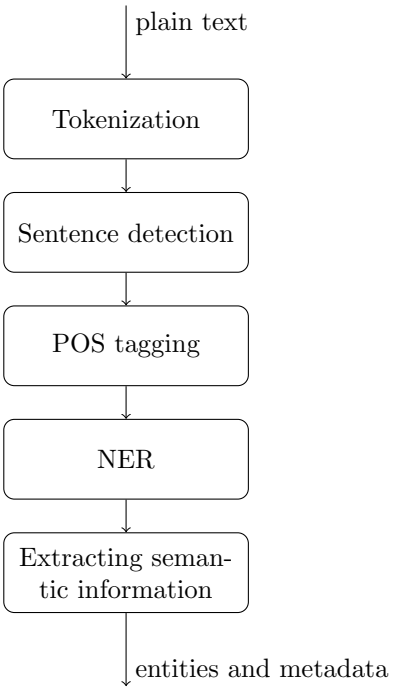

Figure 1.1: The figure shows an overview of the pipeline architecture.

The format of the input and output from each step is very much inspired by the format used in CoNLL with one token per line and the features of each token separated by a tab. Table 1.1 shows an example of the output format used in CoNLL 2002.

Token	Tag
Wolff	B-PER
,	O
currently	O
a	O
journalist	O
in	O
Argentina	B-LOC
,	O
played	O
with	O
Del	B-PER
Bosque	I-PER
in	O
the	O
final	O
years	O
of	O
the	O
seventies	O
in	O
Real	B-ORG
Madrid	I-ORG
.	O

Table 1.1: The table shows an example of the output data in the shared task of CoNLL 2002. The identified entities are: Wolff, Argentina, Del Bosque, and Real Madrid.

Chapter 2

Linguistic background

In this chapter, we introduce the concepts part-of-speech tagging, named entity recognition, and semantic networks. A presentation of those topics has been written by Ingersoll et al. (2012). They also include theory on tokenization and sentence detection. We also describe semantic networks, evaluation methods and the Stockholm-Umeå Corpus in this chapter.

2.1 Part-of-speech tagging

Part-of-speech tagging is the task of deciding what part of speech a token in a text has. There are a number of different parts of speech that a word can have. Some of the common ones include nouns, verbs, adjectives, and prepositions. For a complete list of the different parts of speech used in this master's thesis, see Table A.1 in Appendix A.1.

In Swedish, a word can be inflected in different ways depending on the part of speech. For example if the word is a noun it can be neuter or uter, singular or plural, indefinite or definite, and nominative or genitive. Such features are called morphological features. For example, the Swedish word *bilens* 'the car's' has the following morphological features: uter, singular, definite, and genitive. What kind of morphological features a word can have differs between different parts of speech and some parts of speech cannot have any morphological features at all. Table 2.1 shows an example sentence with the corresponding parts of speech and morphological features for each token.

Word	Part of speech	Morphological features
Bilens	Noun	Uter, singular, definite, genitive
färg	Noun	Uter, singular, indefinite, nominative
är	Verb	Present, active
röd	Adjective	Positive, uter, singular, indefinite, nominative
.	Delimiter	-

Table 2.1: The table shows an example sentence with part of speech and morphological features for each word. In English, the sentence translates to: The car's color is red.

Part-of-speech tagging can be done in a number of different ways, most notable by writing rules or using machine-learning techniques. One thing that has to be decided is the set of tags to be used as the possible parts of speech. For Swedish there are three rather common tagsets. There are the tagsets used in SUC, PAROLE[1], and Granska. The tagset used in SUC and PAROLE can be directly mapped to each other, while the tagset used in the Granska tagger uses an extension of the tagset in SUC. How well a part-of-speech tagger performs depends on the tagset used, among other things.

2.2 Named entity recognition

Named entity recognition is the task of finding named entities in texts. Named entities might include persons, locations, and organizations. The parts of speech of words play an important role in named entity recognition, since a lot of them are proper nouns. Thus, a part-of-speech tagger can be a very useful tool in finding named entities. Common nouns might also be part of named entities, such as the White House. Where House is a common noun and White is an adjective.

If all the named entities that are interesting are known beforehand, the task is much simpler than the case where new (not seen before) entities are to be found. To find such entities, a dictionary of named entities will not suffice.

The task of finding named entities can be divided into three different categories, depending on the type of approach used. The first approach is to write rules manually and the second is to use statistical classifiers. A third approach is to use a dictionary of all the entities one is interested in finding. It is of course also possible to create a system that combines these methods.

Often it is not only interesting to know whether or not a set of words is a named entity, but also what kind of entity it is. This can be a rather hard task since it is rather common for an entity to have more than one possible meaning. If the word Washington has been found for example, it might refer to a city, a person, a state, or possibly something else.

2.2.1 Rule-based techniques

Using a rule-based approach to named entity recognition, information about the current token and its surrounding is used to determine if the token is part of an entity or not. The part of speech is of particular interest since it is common that a proper noun is part of a named entity. Other interesting properties include: initial capitalization, the token is contained in a dictionary of known entities, and morphological features of the token.

Manually written rules have a tendency to be hard to maintain and rather inflexible.

2.2.2 Statistical classifiers

A more robust approach to named entity recognition is to create a statistical model that decides whether a token is part of a named entity or not, using a set of features. Assume there is a function f that will return the type of a token

[1] http://spraakbanken.gu.se/parole/

given a set of features x. What a statistical classifier then does is to approximate the function f with another function h, also taking x as argument. To create such a model (or function), a rather large annotated text is usually used. One of the greatest challenges when creating the model is to choose what features to use. It is rather common that features are dependent on each other. This means that a feature cannot be classified as good or bad, without testing it with different combinations of other features.

2.2.3 Dictionary lookup

If all of the possible named entities one is interested in finding are known beforehand, a simple search for them can be performed. Due to the ambiguity inherent in natural language, this might give rise to false positives. To accommodate for this, the part of speech of a token can be used to a certain degree. It is for example rather uncommon for a named entity to consist of just a preposition or a determiner. Those tokens can then be filtered out.

2.3 Semantic networks

A semantic network can be seen as a graph, where the nodes are entities and the edges are predicates linking the entities together, forming a network. It is semantic since it is concerned with the meaning of entities and what they mean to each other. All entities and predicates have a unique identifier to distinguish between entities or predicates described by the same word in a natural language.

One common way of storing a semantic network is to use the Resource Description Framework (RDF). Each edge in the semantic network graph has a corresponding RDF triple. Such a triple consists of a subject, an object, and a predicate. The nodes in the graph are either unique identifiers or literal data types. Table 2.2 shows two examples of RDF triples. The unique identifiers are enclosed in angle brackets and literal data types are shown within double quotes. Figure 2.1 shows an example of a semantic network. For more information about

Subject	Predicate	Object
<Denmark>	`hasCapital`	<Copenhagen>
<Copenhagen>	`hasPopulation`	“531199”

Table 2.2: The table shows two example RDF triples, each with a subject, an object, and a predicate.

semantic networks and the RDF, see for example Allemang and Hendler (2011).

2.3.1 YAGO2

YAGO2 (Hoffart et al., 2011a) is a large database with information stored as RDF triples, forming a semantic network. The information has been extracted from Wikipedia articles, Geonames, and the WordNet project. One of the most important predicates for this thesis is the `isCalled` predicate, which as the name suggests gives alternative names to the unique entities in the semantic network.

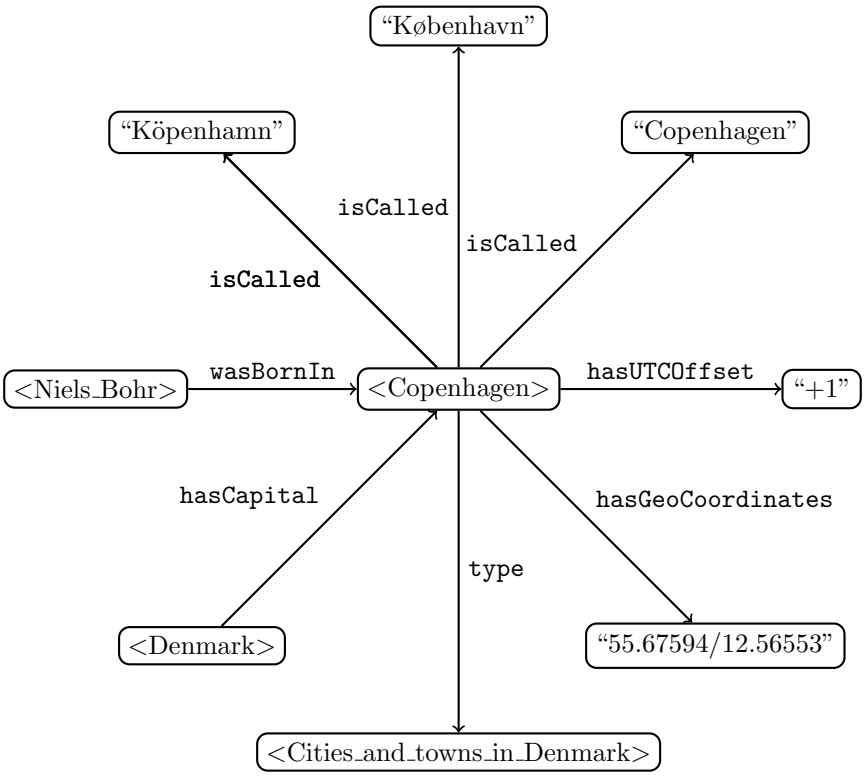

Figure 2.1: In the figure, an example with information from a semantic network is illustrated. Subjects and objects are nodes in the graph and predicates are denoted by arcs from subjects to objects.

2.3.2 Geonames

Geonames is a rather large database containing geographical locations. For each location, there is a list of synonyms for that particular location and its geographical coordinates.

2.3.3 WordNet

WordNet is a database of English words, where each word has been stored with its corresponding semantics. We did not use this part of YAGO2 very much in this project, since the entities that are considered interesting usually is not regular words that can be found in WordNet.

2.3.4 DBPedia

DBPedia is a project in which structured information has been extracted from Wikipedia. Just as in YAGO2, the information is stored as RDF triples. DBPedia is a work in progress and is still being updated.

2.3.5 Database and querying

One common query language used to query databases storing RDF triples is SPARQL (SPARQL Protocol and RDF Query Language). SPARQL is similar to SQL and it is used to get a subset of the information available in a semantic network.

In this thesis, only basic queries are required, answering the questions 'who or what is called a given name?' and 'how is a given entity in the network related to other entities?'. For example, what are the unique identifiers in the semantic network that are called *Köpenhamn*? The result might be the node denoted <Copenhagen> in the center of Figure 2.1. That can then be used to see how Copenhagen is related to other entities in the semantic network. The first SPARQL query for the above given example would be as follows:

```
SELECT ?subject WHERE { ?subject isCalled "Köpenhamn" }
```

The answer to the query would be the following:

<Copenhagen>

The result of the query can then be used to get all predicates and entities for Copenhagen. Figure 2.1 shows some possible predicates and entities related to Copenhagen. For example that Copenhagen is a city or town in Denmark and that Niels Bohr was born there.

2.4 Evaluation methods

When evaluating the performance of information retrieval systems, it is common to measure the precision and recall.

In named entity recognition, there are four possibilities when evaluating found entities and entities that have not been found.

- True positive, an entity that was supposed to be found has been found.
- False negative, an entity that was supposed to be found was not found.
- False positive, no entity was supposed to be found but one was found.
- True negative, no entity was supposed to be found and none was found.

Table 2.3 shows the four listed cases.

	Correct	Not correct
Found	true positive (tp)	false positive (fp)
Not found	false negative (fn)	true negative (tn)

Table 2.3: The table shows the four possible categories of named entities concerning correctness and whether or not they have been found.

The precision is defined as the number of correct entities found divided by the total number of found entities.

$$precision(P) = \frac{correct\ and\ found\ entities}{found\ entities} = \frac{tp}{tp + fp}$$

The recall is defined as the number of correct entities found divided by the total number of correct entities.

$$recall(R) = \frac{correct\ and\ found\ entities}{correct\ entities} = \frac{tp}{tp + fn}$$

There are some problems only using the precision and the recall when evaluating the performance of a system. For example, it is possible to get 100 % recall by simply returning all possible entities. To address this shortcoming, the weighted harmonic mean of the two can be calculated and that is called the F-measure.

$$F = \frac{(\beta^2 + 1)PR}{\beta^2 P + R}$$

Here β is in the interval $[0, \infty]$ and is used for weighting. When set to one, equal weight is given to precision and recall and it is written $F_{\beta=1}$ or F_1.

Another measurement that can be used is the accuracy, in other words the number of correct results both found and not found divided by the total number of results. For named entity recognition, the accuracy is likely to be very high since the set of named entities in a text usually is rather small compared to the whole text. This will result in a high accuracy since the set of true negatives will be much larger than the other three sets in Table 2.3.

$$accuracy = \frac{tp + tn}{tp + fp + fn + tn}$$

For part-of-speech tagging, the accuracy works better as a measurement since every token should be tagged and all tags are equally important. For every token a tag is given and one should be given for every token. Expressed differently, the category of true negatives is empty.

Another common way to show how well a tagger of some sort performs is by using a confusion matrix. In such a matrix, it is possible to see how many of the tokens that were supposed to get a certain tag, got a certain other tag. In other words, it is not only possible to see whether or not a tag is correct but also what kinds of mistakes are most common.

More information about evaluating information retrieval systems can be found in Manning et al. (2008).

2.5 Stockholm-Umeå Corpus 2.0

If machine learning techniques are to be used for part-of-speech tagging and named entity recognition, training data is needed. For Swedish, the options when choosing a corpus to use as training data are rather small. The corpus likely to have been used most when training Swedish part-of-speech taggers is the Stockholm-Umeå Corpus (SUC). SUC is a balanced corpus with texts from a variety of different areas. It contains texts from newspapers, religious texts, and scientific writings to mention a few.

All texts have been annotated with the part of speech and morphological features of all tokens. The different parts of speech and morphological features used in SUC can be found in Appendix A.

Named entities have also been annotated and divided into nine different categories. The categories are: person, animal, myth, place, institution, product, work, event, and other.

For more information about SUC, see the manual (Gustafson-Capková and Hartmann, 2006).

Chapter 3

Analysis of available tools

In this chapter, we analyze the different steps in the pipeline architecture and motivate the use of some tools used. Since it seemed hard to predict how well different approaches would work, we usually carried out simple tests. This meant that for most parts of the developed system, we first implemented a simple solution and if it seemed to be working it was improved.

3.1 Tokenization

The first implementation of the tokenizer split the text at whitespaces and looked for punctuation marks at the end of tokens. We then improved this with a better and easier to maintain tokenizer, inspired by the approach used in the `ClassicTokenizer` of Lucene[1].

3.2 Sentence detection

We also performed sentence detection with a simple approach to start with, basically just looking for tokens only consisting of punctuation marks or having initial capital letters. When we had implemented a more advanced tokenizer, which also categorizes the tokens, we used this to implement a better sentence detector. This made it easy to not allow sentences to contain multiple line feeds for example, commonly used to separate paragraphs.

3.3 Part-of-speech tagging

We have tested all of the three mentioned part-of-speech taggers in Section 1.3. The Granska tagger and Tnt have been evaluated and compared with some other part-of-speech taggers, and the two had the highest accuracy according to Sjöbergh (2003). Since HunPos is a reimplementation of Tnt we expected it to perform similarly. The pre-trained models available for OpenNLP's part-of-speech tagger use a different tagset than SUC, which does not have any morphological features. The Granska tagger does not seem to support UTF-8 as character encoding, which was a problem. This lead us to the use of HunPos

[1] http://lucene.apache.org/

as part-of-speech tagger, although the Granska tagger provides extra features such as a lemmatizer.

3.4 Named entity recognition

Our first approach to NER was to write manual rules looking at the parts of speech of the current and surrounding tokens. We used the following tags:

- B-NE, denoting the start of an entity
- I-NE, denoting the continuation of an entity
- O, denoting a token that is not part of an entity

This differs from some of the available systems and also from the tags used in the shared tasks of CoNLL 2002 and 2003. In those tasks, different categories of named entities were to be found. The categories were persons, locations, organizations, and miscellaneous. Instead of categorizing the entities at the recognition step, we do the disambiguation at a later step; where more information is available via a semantic network. In order to get ideas of what kind of rules might be good, we used example texts from a Swedish newspaper. We did not use SUC to develop the rules, since it would be used to evaluate the performance later. It was thus necessary to avoid peeking. We then improved the NER by using machine learning techniques; we used both logistic regression and support vector machine approaches. The features we selected were inspired by some of the top performing systems in CoNLL 2002 and 2003 (Carreras et al., 2002; Wu et al., 2002; Florian et al., 2003). We tried a small number of different feature combinations and the final system uses the best combination of the ones tested; see Section 4.4.2 for the final selection.

3.5 Entity mapping

In our first approach to mapping an entity to a unique identifier, we simply tried to match the entity to an identifier with the predicate `isCalled`. We then made two improvements in order to get the final system. The first improvement was to convert every object in the `isCalled` predicate to lower case, as well as the entities that were to be mapped. To be able to map entities written as genitive with an s at the end, we took the morphological features of the named entities into account at the mapping step.

Chapter 4

Architecture overview

In this chapter, we present implementation details of the different steps in the pipeline architecture. We use plain text as input to the program and we output metadata in the form of semantic information about the found entities in the input text. Figure 4.1 shows all the steps in the developed program. Figure 4.2 shows a screenshot of the graphical user interface that was developed in order to help visualize the result from the different steps.

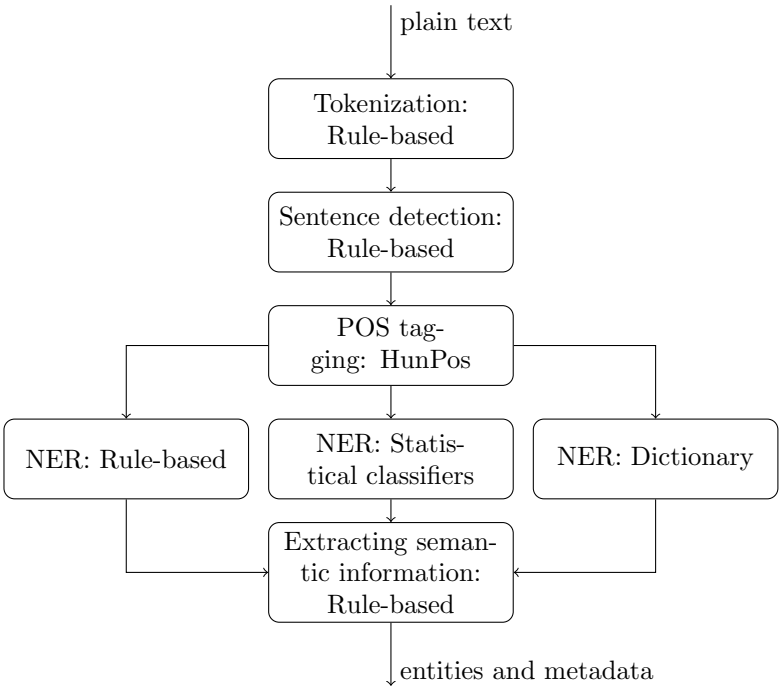

Figure 4.1: The figure shows all steps taken in processing an input text.

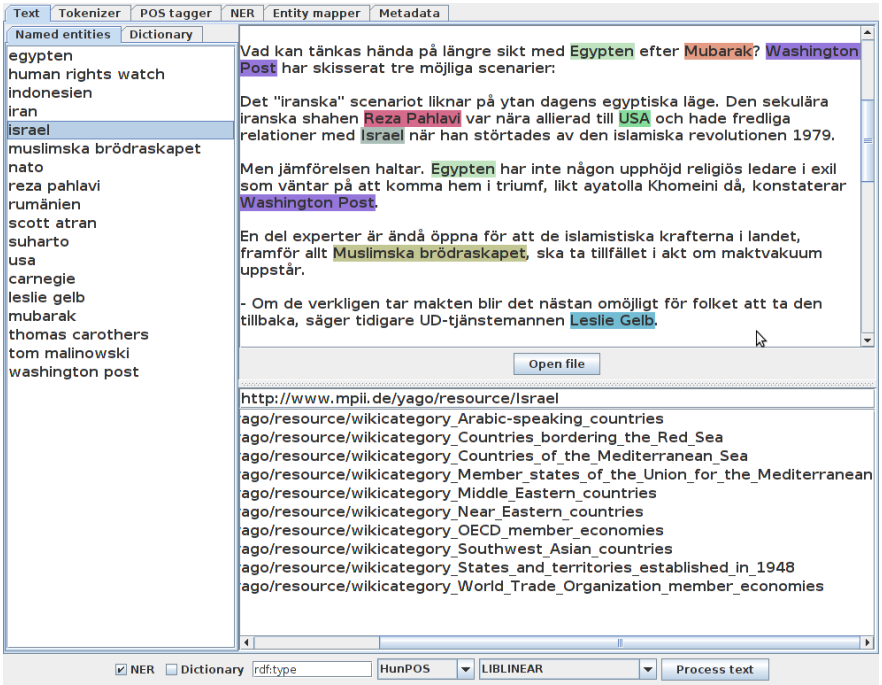

Figure 4.2: This is a screenshot of the graphical user interface used to visualize the results from the different steps in the program. The left pane shows all the found entities and they are also highlighted in the main pane. In the bottom pane, is the unique identifier for Israel together with some extracted information from the semantic network about Israel.

4.1 Tokenization

We have used parts of Lucene to tokenize the text. Since the `ClassicTokenizer` in Lucene removes punctuation marks and some stop words, we have used a modified version of it. Just as in the `ClassicTokenizer`, rules define what a token should look like and then JFlex[1] is used to generate a Java file representing a finite state automaton. The output from this step is a list of tokens and their type.

Our tokenizer does not only find tokens used as input to the part-of-speech tagger; it also recognizes tokens consisting of multiple line feeds. We use this in the next step to make the sentence detection better. Figure 4.3 shows an example sentence that has been tokenized.

| Oppositionen | i | Italien | rasar | över | Silvio | Berlusconis | ord | . |

Figure 4.3: The table shows an example sentence that has been tokenized.

[1] http://jflex.de/

4.2 Sentence detection

In this step, we divide the tokens into lists forming sentences. We do this using the type of the tokens and the token itself. The most interesting tokens are the ones consisting of one or more of the punctuation marks: ., ?, !, and :. Tokens consisting of multiple line feeds always split sentences. This is useful for detecting headings that should be detected as sentences, although they rarely end with a punctuation mark. When the tokens have been processed, the whole text is output with one token per line and each sentence separated with an empty line.

4.3 Part-of-speech tagging

In this step, we tag each token in the sentences with its part of speech. Depending on the part-of-speech tag and the word itself, morphological features might also be added. We used HunPos as tagger and the Stockholm-Umeå Corpus version 2.0 as training data. Table 4.1 shows an example sentence with output from HunPos, including parts of speech and morphological features for all tokens.

Token	POS	Morphological features
Oppositionen	NN	UTR,SIN,DEF,NOM
i	PP	-
Italien	PM	NOM
rasar	VB	PRS,AKT
över	PP	-
Silvio	PM	NOM
Berlusconis	PM	GEN
ord	NN	NEU,PLU,IND,NOM
.	MAD	-

Table 4.1: The table shows an example sentence where each token has been tagged with its corresponding part of speech and morphological features.

4.4 Named entity recognition

We have used three different approaches to identify named entities in the input texts. The approaches are: rule-based, statistical classifiers, and dictionary lookup.

In this step, we add another tag to each token in order to identify named entities. The tagset used follow the same standard as the tags in CoNLL 2002, namely B, I, and O. B is used to mark the beginning of an entity, I to tag a token as being inside an entity, and O is used for tokens that are not part of an entity. Those letters are then followed by a dash and then a category. Table 4.2 shows a tagged example sentence without any categories.

Oppositionen	i	Italien	rasar	över	Silvio	Berlusconis	ord	.
O	O	B-NE	O	O	B-NE	I-NE	O	O

Table 4.2: The table shows an example sentence where each token has been tagged with a named entity tag. The found entities are *Italien* and *Silvio Berlusconis*.

4.4.1 Manually-written rules

In this approach, we used the part-of-speech tag and the morphological features of the current token and the two surrounding tokens to decide if the token is at the beginning of a named entity, inside one, or outside and not part of any named entity. One part-of-speech tag that is of particular interest is the proper noun tag since that is the most common tag for a named entity, other nouns and foreign words are also rather common.

4.4.2 Statistical classifiers

We have used both LIBLINEAR (Fan et al., 2008) and LIBSVM (Chang and Lin, 2011) as statistical classifiers. Both LIBLINEAR and LIBSVM are general machine learning tools in the sense that they can be used classify other things that does not involve natural language. LIBLINEAR uses logistic regression and linear support vector machine techniques while LIBSVM uses other support vector machine techniques. We used the Stockholm-Umeå Corpus version 2.0 as training data. After a conversion from the format used by SUC 2.0 to the format used in the shared task of CoNLL 2002, the corpus was used when training both LIBLINEAR and LIBSVM.

The set of features used to train the classifiers were inspired by some of the top performing systems in CoNLL 2002 and CoNLL 2003. The features used for LIBLINEAR were:

1. The part-of-speech tag of the current token, the previous two, and the one after
2. The current token, the previous two, and the one after
3. The named entity tags of the previous two tokens
4. Initial capital letter and the first token of a sentence
5. Initial capital letter and not the first token of a sentence
6. All letters in the token capitalized
7. The current token containing one or more periods
8. The current token is a single character

When the current token is the first, second, or last token of a sentence, there is a problem with finding features for previous or following tokens. We solved this by introducing a special feature for those tokens, simply saying that no token exists before or after.

We did not use the second enumerated feature above with LIBSVM. We tried, but it resulted in all tokens being tagged as outside.

Table 4.3 shows an example where the token *Silvio* is about to be tagged. The corresponding features and their values are shown in Table 4.4. The token features only have a value if the token exists in the training data.

Index	Token	POS	Named entity tag
$i-5$	Oppositionen	NN	O
$i-4$	i	PP	O
$i-3$	Italien	PM	B-NE
$i-2$	rasar	VB	O
$i-1$	över	PP	O
i	Silvio	PM	?
$i+1$	Berlusconis	PM	?
$i+2$	ord	NN	?
$i+3$	.	MAD	?

Table 4.3: The table shows a situation where the token *Silvio* is to be tagged.

Feature	Value
POS_{i-2}	VB
POS_{i-1}	PP
POS_{i}	PM
POS_{i+1}	PM
Token_{i-2}	rasar
Token_{i-1}	över
Token_{i}	Silvio
Token_{i+1}	Berlusconis
NE_{i-2}	O
NE_{i-1}	O
4	False
5	True
6	False
7	False
8	False

Table 4.4: The table shows all the features and corresponding values for the tagging example in Table 4.4.

4.4.3 Dictionary lookup

The dictionary we have used in this part is Swedish MeSH[2], an extension of MeSH[3] with Swedish synonyms for the terms found in MeSH. MeSH is short for Medical Subject Headings and, as the name suggests, contains medical terms.

[2]http://mesh.kib.ki.se/swemesh/swemesh_en.cfm
[3]http://www.nlm.nih.gov/mesh/meshhome.html

It would be possible to do a simple search in the text for all the medical terms. This would give some results that are not correct, due to the ambiguity present in natural language. For example, the Swedish medical term *genom* 'genome' is also a common preposition in Swedish, which of course should not be found as a named entity. To accommodate for this shortcoming, we used the part-of-speech tags of the tokens to filter out unwanted results.

Since the found entities should be mapped to a unique identifier and are known before they are being searched for in the texts, we decided the unique identifiers beforehand. In MeSH, all terms have a unique id number and this can also be found in some of the info boxes in Wikipedia articles. We used this to get an exact mapping for some medical terms to a unique identifier in the semantic network. In the other cases, we used the medical terms themselves to get a mapping.

There were also some very common terms available that are likely to occur in a lot of documents and thus are not that interesting. To filter out these terms, we calculated the inverse document frequency (idf) for all terms in the given document set. We then removed every term below a certain threshold. The inverted document frequency is a measurement of how common a term is in a predefined set of documents. A higher idf for a term means that it is rarer compared to other terms in the same set of documents. Equation 4.1 shows how the idfs are calculated. t is the current term, N is the number of documents in the collection, and df_t is the document frequency of term t.

$$idf(t) = 1 + \log \frac{N}{df_t + 1} \tag{4.1}$$

For more information about inverse document frequency, see Manning et al. (2008).

With the filtered dictionary of Swedish synonyms, mapped to unique YAGO2 entities, we used Lucene to do a phrase query for all the terms in the dictionary. If a term was found, we tagged the token or tokens in the same way as in the other two approaches.

4.5 Extracting semantic information

In this step, we find semantic information about the entities that have been found in the previous steps. We achieved this by querying the database, using SPARQL, for entities with the predicate `isCalled` matching the found named entity. We also take the morphological features of the tokens in the named entities into account here. For example, proper nouns with morphological feature genitive are of particular interest. Words that have the morphological feature genitive in Swedish, always ends with an s. An s is not always added to the end of the word though, since it might end with an s in its nominative form. To find the entity in the semantic network, it would be good to have the lemma of the entity. If the found entity is in genitive, the s cannot simply be removed to find the lemma. This is unless the word ends with a colon and then an s, in that case there is no ambiguity and those characters can be removed in order to retrieve the lemma. This stage of the program could clearly benefit from a proper lemmatizer.

In the implementation we remove the s if the token has been tagged as genitive. If that does not give any results, we query the database with no modifications to the named entity. Table 4.5 shows two example entities and their corresponding unique identifiers.

Named entity	YAGO2 id
Italien	http://www.mpii.de/yago/resource/Italy
Silvio Berlusconis	http://www.mpii.de/yago/resource/Silvio_Berlusconi

Table 4.5: The table shows two named entities and their corresponding unique identifiers in YAGO2.

4.5.1 Database

We have used Sesame from openRDF.org[4] to store all the information in the semantic network. It is a free open-source project written in Java and it supports two query languages, SeRQL and SPARQL. We made some modifications to YAGO2 before the entries were added to the database. YAGO2 has a lot of information available in languages that does not occur in Swedish texts, for example what an entity is called in Chinese. Thus, we filtered out all RDF triples containing characters that do not occur in Swedish texts. We also converted all objects in the `isCalled` predicate to lower case, to easier find entities with odd capitalization patterns.

4.5.2 Disambiguation

For some of the found named entities, more than one match is found using the `isCalled` predicate in the semantic network. Thus, some sort of disambiguation is needed in order to decide which unique identifier, if any of the found ones, is the right one. For this, we have used a very simple heuristic. In the current implementation, we choose the entity with the most related information available. Obviously this is far from correct in all situations, but it is a baseline to improve upon.

[4] http://www.openrdf.org/

Chapter 5

Results

In this chapter, we report the results of the different parts of the system. However, we have not evaluated how well the tokenizer and the sentence detector perform. The reason is that there is no test data available. Those tasks are also much simpler than the other tasks and they are likely to perform well. To be able to measure the performance accurately, we would also require test data with enough diversity; since both the tokenizer and the sentence detector perform almost perfectly on the news articles. We have not made any measurements on the NER part using a dictionary since it will always find the terms in the dictionary.

5.1 Part-of-speech tagging

The performance of HunPos trained on SUC has already been evaluated by Eva Forsbom at Uppsala University[1]. With a 10-fold cross-validation the accuracy is 95.47 ± 0.15, a comparison with other tagsets is also available for the interested reader.

5.2 Named entity recognition

We have evaluated the performances of the different NER approaches using SUC. To start with we have measured how many entities there are in SUC and what categories they belong to. The result of that measurement is shown in Table 5.1.

We have measured the number of found entities and the number of correct entities in all texts in SUC with the rule-based approach. Using those figures we have also calculated the precision, recall, and F-measure; the figures are shown in Table 5.2.

To measure the performance of the statistical approach using LIBLINEAR, we used a 10-fold cross-validation on SUC. The only parameter we changed was the solver type; all other parameters were set to their respective default values. We used the same fold division in each of the eight evaluations. The solver types that exist in LIBLINEAR are enumerated below.

[1] http://stp.lingfil.uu.se/ evafo/resources/taggermodels/HunPosmodels.html

Category	Count
Animal	364
Event	245
Institution	6332
Myth	280
Other	542
Person	15128
Place	8773
Product	638
Work of art	1887
Total	34189

Table 5.1: The table shows the number of named entities occurring in the different categories in SUC.

Exist	Found	Correct	Precision (%)	Recall (%)	F_1 (%)
34189	32943	30010	91.10	87.78	89.41

Table 5.2: The table shows how many of the entities in SUC that were found and correct by the handwritten rules. The precision, recall, and F-measure are also shown.

1. L2-regularized logistic regression (primal)
2. L2-regularized L2-loss support vector classification (dual)
3. L2-regularized L2-loss support vector classification (primal)
4. L2-regularized L1-loss support vector classification (dual)
5. Multi-class support vector classification by Crammer and Singer
6. L1-regularized L2-loss support vector classification
7. L1-regularized logistic regression
8. L2-regularized logistic regression (dual)

For each solver type, we measured the number of found entities and how many of those were correct. We also calculated the precision, recall, and F-measure and the results are shown in Table 5.3.

One other thing that might be interesting to see is what kind of tagging mistakes that are most common. Thus, we calculated the confusion matrix shown in Table 5.4; using the solver type with the best F-measure (91.35).

To be able to make a better comparison with the results from the shared task of CoNLL 2002 and 2003, we also measured the performance using the same categories as in those competitions. One problem is that the named entities in SUC are not divided into the same categories as those used in the shared task of CoNLL 2002 and 2003. We solved this by mapping the categories in SUC to the wanted ones in the following way. The person category in SUC was no problem since person was one of the four categories wanted. Place was mapped

Solver type	Found entities	Correct entities	Precision (%)	Recall (%)	F_1 (%)
1	33132	30587	92.32	89.46	90.87
2	33372	30852	92.45	90.24	91.33
3	33228	30792	**92.67**	90.06	**91.35**
4	33317	30826	92.52	90.16	91.33
5	**33601**	**30889**	91.93	**90.35**	91.13
6	33032	30533	92.43	89.31	90.84
7	33047	30614	92.64	89.54	91.06
8	33190	30639	92.31	89.62	90.95

Table 5.3: The table shows the NER result of using the different solver types in LIBLINEAR. A 10-fold cross-validation was used on all texts in SUC.

	B-NE (A)	I-NE (A)	O (A)
B-NE (P)	31690	622	916
I-NE (P)	241	10164	363
O (P)	2258	2329	1118173

Table 5.4: The table shows a confusion matrix for the named entity recognition tags. The predicted tags (P) are in the left column and the actual tags (A) are in the top row.

to location and institution was mapped to organization while the rest (animal, myth, product, work, event, and other) were mapped to miscellaneous. We used LIBLINEAR in the same way as before with the solver type L2-regularized L2-loss support vector classification (primal), the results are shown in Table 5.5.

Category	Exist	Found	Correct	Precision (%)	Recall (%)	F_1 (%)
Persons	15128	17198	13626	79.23	90.07	84.30
Organizations	6332	4540	3089	68.04	48.78	56.82
Locations	8773	8974	6926	77.18	78.95	78.05
Miscellaneous	3956	2051	1249	60.90	31.57	41.58
Total	34189	32763	24890	75.97	72.80	74.35

Table 5.5: The table shows how many entities that exist in each category, how many that were found, and how many that were correct. The precision, recall, and F-measure were calculated for the NER using LIBLINEAR.

We have evaluated the performance of LIBSVM in a similar way as LIBLINEAR has been evaluated. One problem with LIBSVM is that it is much slower than LIBLINEAR, both in training and predicting. For that reason, we have only used a subset of 50 texts from SUC in a 10-fold cross-validation. In the evaluation, all parameters were set to their default values and we did not test any other configuration. In order to compare it with LIBLINEAR, we ran LIBLINEAR on the same test data with the solver type L2-regularized L2-loss

support vector classification (primal) and all other parameters set to default. The results from both LIBSVM and LIBLINEAR are shown in Table 5.6. It is important to notice that we did not use the same features for LIBSVM as for LIBLINEAR, due to the problem mentioned in Section 4.4.2.

Classifier	Found	Correct	Precision (%)	Recall (%)	F_1 (%)
LIBSVM	6189	5788	93.52	89.83	91.64
LIBLINEAR	6285	5923	94.24	91.93	93.07

Table 5.6: The table shows the number of found entities and correct entities using LIBSVM and LIBLINEAR on 50 texts from SUC. LIBLINEAR was used with solver type L2-regularized L2-loss support vector classification (primal). The total number of entities in the 50 texts was 6443. The table also shows the precision, recall, and F-measure.

5.3 Mapping with the semantic network

In order to measure the performance of the mapping of entities to unique identifiers in the semantic network, we used SUC. We have calculated the number of unique phrases annotated as named entities in the 500 texts of SUC, those are 13392. Note that this is unique phrases and not unique entities, since two different phrases might refer to one entity. We then tried to find a corresponding unique identifier for each phrase in the semantic network. We managed to map 5121 of the 13392 phrases to identifiers in the semantic network. Of those 5121 phrases, we found 4458 unique identifiers in the semantic network. Note that the found identifiers might not necessarily be the correct ones in all contexts, or even in any of the contexts the entity appears in.

We also wanted to measure how well the mapping worked for a whole text. We picked five texts in SUC and annotated the named entities in them with their corresponding unique identifier in the semantic network, if such an identifier existed. For each text, we then got a set of wanted unique identifiers. This set was then compared to the output set of the program, see Table 5.7.

Exist	Found	Correct	Precision	Recall	F_1
169	158	132	83.54	78.11	80.73

Table 5.7: The table shows the number of existing, found, and correct unique identifiers in five texts from SUC. The table also shows the precision, recall, and F-measure.

5.4 Execution time

We have measured the average execution time for all the processing steps by running each step 10 times and taking the average. The test data we used was a news article with 210 tokens, 9 sentences, 19 found named entities, and 11

found unique identifiers. The resulting average times for all the steps can be found in Table 5.8.

Processing step	Execution time (ms)
Tokenization	2.1
Sentence detection	2.0
Part-of-speech tagging	318.6
NER (Rule-based)	19.5
NER (LIBLINEAR)	16.3
NER (LIBSVM)	251.6
NER (Dictionary)	10540.7
Extracting semantic information	74.8

Table 5.8: The table shows the average execution time of all the processing steps when processing a news article.

Chapter 6

Discussion

In this thesis, we have presented a system that is intended to be used to create a richer search experience. We have implemented named entity recognition to find named entities and mapping of those entities to semantic networks. In this chapter, we discuss the results, possible improvements, give conclusions, and suggest future work.

6.1 Results and improvements discussion

We have not measured the performance of the tokenization and sentence detection steps. The reason for this is that the texts used as input are not diverse enough, meaning that it is likely to show perfect or almost perfect performance on those texts. One of the reasons for measuring the performance of the steps individually, is to get a hint of which step is likely to be easiest to improve upon. We believe that the performance of both the tokenization and the sentence detection would be the two best performing steps, since they are the first two and they are rather easy compared to the other tasks. Thus, the performance figures would not be very useful for the previously mentioned task.

Using a larger corpus as training data, we believe that the performance of HunPos can be improved. Another possible way to improve the performance when the input texts come from a specific domain would be to only use texts from the same domain as training data, news articles or medical texts in this case. This could be useful since the accuracy usually is higher for words that exist in the training data. Both these improvements rely on the creation or extension of an annotated corpus, a task that can be very time-consuming. It is also a trade-off between having good data but a small quantity and having worse data in a larger quantity.

Another option to consider might be to use a different part-of-speech tagger, the most obvious option for Swedish being the Granska tagger. Of course, then the encoding issue has to be solved. On the other hand, there would no longer be any need for a tokenizer nor a sentence detector, since those are built into the Granska tagger.

We believe a third improvement of the part-of-speech tagger could be to use a different tagset. This could for example be the tagset used by Granska, which according to Carlberger and Kann (1999) gives a slight improvement compared

to the SUC tagset.

With more time we could probably have improved the rule-based named entity recognition, since the rules are rather simple. They only use the part of speech tags, the previous named entity tag, and the morphological features. It would be possible to also use some of the other features used in the statistical approaches. One reason why we did not do this, is that handwritten rules tend to be harder to maintain compared to features used in machine learning. When developing rules it is also important to not look at the evaluation data.

We also believe it to be possible to improve the performance of the two statistical approaches. We have not fully explored all possible combinations of the features. It is also possible to encode the same features in different ways and that might lead to slightly different results.

Another way to optimize the performance of the named entity recognition could be to change the parameters used by LIBLINEAR and LIBSVM. We have tested all solver types for LIBLINEAR and there are some minor differences in the performance, see Table 5.3. We chose the solver type L2-regularized L2-loss support vector classification (primal) for the final system because it showed best performance in the evaluation. We did not try any other values than the default ones for the other parameters. For LIBSVM, we only tried the default parameters. Thus, we believe it might be possible to improve the results in Table 5.3 and Table 5.6.

We wanted to compare the performance of the rule-based NER, LIBLINEAR, and LIBSVM. Table 5.6 shows that LIBLINEAR performs better than LIBSVM on the used test set. Comparing the figures in Table 5.2 to the figures in Table 5.3, we see that LIBLINEAR performs better than our rules. It is important to notice that we have evaluated the two approaches in slightly different ways. For the rule-based approach, we used all texts in SUC and with LIBLINEAR we used all texts in a 10-fold cross-validation. The statistical approaches are also more flexible, since it is much easier to change the tagset used compared to the rule-based approach.

The figures in Table 5.5 can be compared to the performance of the systems competing in the shared tasks of CoNLL 2002 and 2003. Those systems reached F-measures between 47.74 % and 88.76 %, compared to 74.35 %. This suggests that the developed system at least have some potential. There are however some problems with this comparison. It does not use the same test set and they are not even in the same language.

In the confusion matrix in Table 5.4, it can be seen that the two most common mistakes are to tag beginnings and continuations of entities as outside. We believe one possible explanation for this might be related to what has been annotated as a named entity in SUC. A particularly bad example is *Förre biträdande försvarsminister Richard Perle*. In that example, it would probably have been better if only *Richard Perle* would have been annotated as a named entity. Even if such an entity were to be found, we believe it also would make the mapping to unique identifiers in the semantic network harder.

Although the mapping from a found entity to a unique identifier in the semantic network works rather well, there is at least one problem with it. It will always map the same entity to the same unique identifier in the semantic network; this is regardless of the context in which the entity appears. For example, George Bush will always be mapped to the YAGO2 identifier <George_W._Bush>; another possible option would be his father with the same name but with the

YAGO2 identifier <George_H._W._Bush>. It is also important to notice that we have only considered entities that exist in YAGO2 in the evaluation shown in Table 5.7. It is rather obvious that the disambiguation can be improved, for example using the approach taken by Hoffart et al. (2011b).

The test set used to get the results in Table 5.7 is a bit too small to draw conclusions. The reason for us not using a larger set is that no test data exist, so we have annotated the used texts by hand.

Looking at the execution times of the different processing steps in Table 5.8, it can be seen that that POS tagging, NER with LIBSVM, and NER with a dictionary are the slowest steps. We believe it would be possible to do the POS tagging a bit faster if a tagger implemented in Java could be used. The execution time of LIBSVM is hard to do something about; on the other hand the dictionary approach could easily be improved a lot. The current implementation is not very good and there are other implementations that should be rather easy to implement and much faster.

6.2 Conclusions

The results show that the approach taken in this thesis to extract structured information can be useful. Statistical approaches to NER are preferable if training data is available. From the use in this thesis, it also looks as if LIBLINEAR outperforms LIBSVM both when comparing F-measure and processing speed. The information available in semantic networks can be very useful in extending named entity recognition systems.

6.3 Future work

None of the steps are perfect, so there is room for improvement everywhere. The disambiguation of a named entity, when there are multiple possible identifiers in the semantic network, is something that we believe could easily be improved. This step of the system could benefit from a lemmatizer. We also think that using a coherence graph (Hoffart et al., 2011b) would be an interesting approach.

Another obvious thing to make better is the dictionary lookup of medical terms. We believe it should be rather simple to implement a much faster algorithm for finding the terms.

In news articles, it is rather common for persons to occur multiple times. One common thing is to mention the full name of the person the first time and then only one of the names (first or last name). This can be problematic since the different occurrences might be mapped to different identifiers in the semantic network. One way to try and solve this would be to create a system for solving coreferences.

The statistical NER approaches could possibly be improved by using a different tagset. This has been suggested by Ratinov and Roth (2009) and they use five different tags instead of three. The tags suggested are beginning, inside, last token of multi-token chunk, unit-length chunk, and outside.

Appendix A

SUC 2.0 tags

A.1 POS tags used in SUC 2.0

Swedish part of speech	English translation
Adverb (AB)	Adverb
Determinerare (DT)	Determiner
Frågande/relativt adverb (HA)	Interrogate/Relative Adverb
Frågande/relativ determinerare (HD)	Interrogate/Relative Determiner
Frågande/relativt pronomen (HP)	Interrogate/Relative Pronoun
Frågande/relativt possessivt pronomen (HS)	Interrogate/Relative Possessive
Infinitivmärke (IE)	Infinitive Marker
Interjektion (IN)	Interjection
Adjektiv (JJ)	Adjective
Konjunktion (KN)	Conjunction
Substantiv (NN)	Noun
Particip (PC)	Participle
Partikel (PL)	Particle
Egennamn (PM)	Proper Noun
Pronomen (PN)	Pronoun
Preposition (PP)	Preposition
Possessivt pronomen (PS)	Possessive
Grundtal (RG)	Cardinal number
Ordningstal (RO)	Ordinal number
Subjunktion (SN)	Subjunction
Utländskt ord (UO)	Foreign Word
Verb (VB)	Verb
Större skiljetecken (MAD)	Major delimiter
Mindre skiljetecken (MID)	Minor delimiter
Parvis skiljetecken (PAD)	Pairwise delimiter

Table A.1: The table shows all the different parts of speech used in SUC 2.0.

A.2 Morphological features used in SUC 2.0

Feature	Value	POS where feature applies
Gender	Uter (UTR)	DT, HD, HP, JJ, NN, PC, PN, PS, (RG, RO)
	Neuter (NEU)	
	Masculine (MAS)	
Number	Singular (SIN)	DT, HD, HP, JJ, NN, PC, PN, PS, (RG, RO)
	Plural (PLU)	
Definiteness	Indefinite (IND)	DT, (HD, HP, HS), JJ, NN, PC, PN, (PS, RG, RO)
	Definite (DEF)	
Case	Nominative (NOM)	JJ, NN, PC, PM, (RG, RO)
	Genitive (GEN)	
Tense	Present (PRS)	VB
	Preterite (PRT)	
	Supinum (SUP)	
	Infinitive (INF)	
Voice	Active (AKT)	VB
	S-form (SFO)	
Mood	Subjunctive (KON)	VB
Participle form	Present (PRS)	PC
	Perfect (PRF)	
Degree	Positive (POS)	(AB), JJ
	Comparative (KOM)	
	Superlative (SUV)	
Pronoun form	Subject form (SUB)	PN
	Object form (OBJ)	
	Compound (SMS)	All
	Abbreviation (AN)	

Table A.2: The table shows the different morphological features used in SUC 2.0 and to which parts of speech they apply.

Bibliography

Dean Allemang and James Hendler. *Semantic Web for the Working Ontologist: Effective Modeling in RDFS and OWL*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2nd edition, 2011. ISBN 0123859654, 9780123859655.

Thorsten Brants. Tnt - a statistical part-of-speech tagger. In *Proceedings of the Sixth Applied Natural Language Processing (ANLP-2000)*, Seattle, WA, 2000.

Johan Carlberger and Viggo Kann. Implementing an efficient part-of-speech tagger. *Softw. Pract. Exper.*, 29(9):815–832, July 1999. ISSN 0038-0644.

Xavier Carreras, Lluís Màrques, and Lluís Padró. Named entity extraction using adaboost. In *Proceedings of CoNLL-2002*, pages 167–170. Taipei, Taiwan, 2002.

Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011.

Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008.

Jenny Rose Finkel, Trond Grenager, and Christopher Manning. Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL '05, pages 363–370, Stroudsburg, PA, USA, 2005. Association for Computational Linguistics.

Radu Florian, Abe Ittycheriah, Hongyan Jing, and Tong Zhang. Named entity recognition through classifier combination. In Walter Daelemans and Miles Osborne, editors, *Proceedings of CoNLL-2003*, pages 168–171. Edmonton, Canada, 2003.

Sofia Gustafson-Capková and Britt Hartmann. Manual of the stockholm umeå corpus version 2.0, December 2006.

Péter Halácsy, András Kornai, and Csaba Oravecz. HunPos: an open source trigram tagger. In *Proceedings of the ACL 2007 Demo and Poster Sessions*, pages 209–212, Stroudsburg, PA, USA, 2007. Association for Computational Linguistics.

Johannes Hoffart, Fabian M. Suchanek, Klaus Berberich, Edwin Lewis-Kelham, Gerard de Melo, and Gerhard Weikum. Yago2: Exploring and querying world knowledge in time, space, context, and many languages. In Sadagopan Srinivasan, Krithi Ramamritham, Arun Kumar, M. P. Ravindra, Elisa Bertino, and Ravi Kumar, editors, *Proceedings of the 20th International Conference Companion on World Wide Web (WWW 2011)*, pages 229–232, Hyderabad, India, 2011a. Association for Computing Machinery (ACM), ACM.

Johannes Hoffart, Mohamed Amir Yosef, Ilaria Bordino, Hagen Fürstenau, Manfred Pinkal, Marc Spaniol, Bilyana Taneva, Stefan Thater, and Gerhard Weikum. Robust disambiguation of named entities in text. In *Conference on Empirical Methods in Natural Language Processing, Edinburgh, Scotland, United Kingdom 2011*, pages 782–792, 2011b.

G.S. Ingersoll, T.S. Morton, and A.L. Farris. *Taming Text: How to Find, Organize, and Manipulate It*. Manning, 2012. ISBN 9781933988382.

Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schtze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008. ISBN 0521865719, 9780521865715.

Lev Ratinov and Dan Roth. Design challenges and misconceptions in named entity recognition. In *CoNLL '09: Proceedings of the Thirteenth Conference on Computational Natural Language Learning*, pages 147–155, Morristown, NJ, USA, 2009. Association for Computational Linguistics.

Jonas Sjöbergh. Combining pos-taggers for improved accuracy on Swedish text. In *Proceedings of NoDaLiDa 2003*, Reykjavik, Iceland, 2003.

Dekai Wu, Grace Ngai, Marine Carpuat, Jeppe Larsen, and Yongsheng Yang. Boosting for named entity recognition. In *Proceedings of CoNLL-2002*, pages 195–198. Taipei, Taiwan, 2002.