MASTER'S THESIS | LUND UNIVERSITY 2016

HERD - Hajen Entity Recognition and Disambiguation

Anton Södergren

Department of Computer Science
Faculty of Engineering LTH

ISSN 1650-2884
LU-CS-EX 2016-21

HERD

(Hajen Entity Recognition and Disambiguation)

Anton Södergren
karl.aj.sodergren@gmail.com

June 13, 2016

Master's thesis work carried out at
the Department of Computer Science, Lund University.

Supervisor: Pierre Nugues, pierre.nugues@cs.lth.se
Examiner: Elin Anna Topp, elin_anna.topp@cs.lth.se

Abstract

This thesis describes the process to build an entity recognizer and disambiguator, named HERD. The goal of the system is to find mentions of entities in text and link those mentions to a unique identifier. This system is designed to be multilingual and has versions in English, French and Swedish.

I use Wikipedia as a knowledge source of both names and concepts, and Wikidata, a language agnostic, structured knowledge source, for unique identifiers. The system collects the links on Wikipedia articles to count and analyze them. The link is seen as a mention, that consists of a label and an address, that the system uses as a name and an identifier. The address is translated into a Wikidata Q-number. When the system parses a new document, each recognized name is linked to a unique identifier.

I have explored logistic regression, PageRank, and feature vectors based on the Wikipedia categories to improve the name recognition, and select the best candidate for each name.

The system is evaluated with the same method as used in the ERD'14 competition, and reached an F1-score of 0.701, which would have placed it 6th, out of 17 competitors, 6 percentage points lower than the highest scoring participant.

Keywords: MSc, ERD, NER, named entities, Wikipedia, Wikidata, Wikification

Acknowledgements

There are several people who have made this thesis possible for me. First of all, I would like to thank my supervisor Pierre Nugues, for the idea, as well as all the advice, contagious enthusiasm and resources he has provided.

I would also like to thank Marcus Klang for his superb technical knowledge, and relentless helpfulness, as well as the other members of the Hajen research group, for rewarding discussions and advice.

The Computer Science department at Lund University also deserves credit for providing me with the resources and tools I needed in order to produce any results at all.

Lastly I would like to thank Sofia Rubertsson, who has been supportive and eager to listen when I needed it. Thank you.

Contents

Chapter 1
Introduction

In order to build artificial intelligence, that is able to answer questions or have a normal conversation, we need to enable computers to understand natural languages, like Swedish or English. Since these are unstructured and ambiguous by nature, they are difficult for a computer to understand. This thesis, and the system it describes, aims to make natural languages a little more structured by finding names in text and linking those names to unique identifiers.

The process of finding names is referred to as **named entity recognition** (NER), and the process of determining who, or what, is meant is called **disambiguation** or **linking**. The question I am trying to answer is: *How can a system that handles entity recognition and disambiguation in a multilingual context be constructed?*

1.1 Problem definition

To understand the difficulties with this project, it is necessary to discuss the definition of a name. Most people have a fairly good understanding of what a name is, along the lines of *a word or phrase to identify a specific person, place or object*. That definition is mostly correct, but misses the point that the identification is weak, since most names could refer to many different things. Take the highly common Swedish personal name *Maria Johansson* as an example. It has more than 11 000 possible meanings. Most things do not have a completely unique name, but given a context; a country, a workplace, a city or a circle of friends, they can most likely be identified with confidence.

1.1.1 Disambiguation

John Williams wrote the music for Jurassic Park, is an example of a possibly ambiguous mention of the very common name *John Williams*. We are, as humans, able to say who is meant through context. The full, non ambiguous, identification is rarely needed (if it even

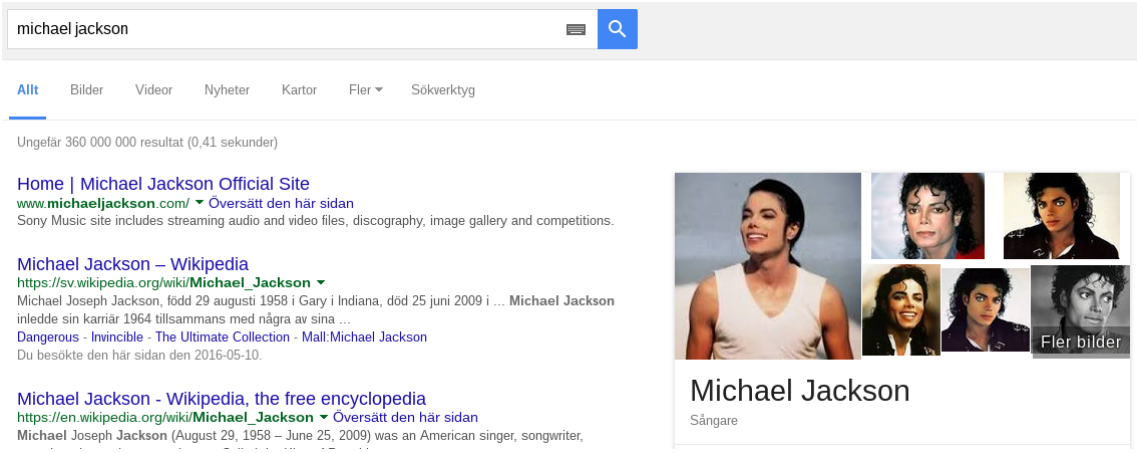

Figure 1.1: An example of a linked named entitiy, by who is the most common.

exists) as we are usually able to conclude what specific object is meant by the surrounding clues. In order to make this example understandable for a computer we need to express context with mathematical models and algorithms, which is not trivial.

1.1.2 Recognition

Recognizing phrases as named entities in the first place is essential to be able to disambiguate the name. This is not as simple as it may seem. We can not fully rely on capitalization since each sentence starts with capital letter, some languages use capital letters for all common nouns, and some names are not meant to be capitalized. We cannot rely on dictionaries of names either, since many names, especially titles of art pieces and names of companies, only consist of commonly used words or phrases. We need to find a more complex approach in order to recognize names with precision.

1.2 Applications

Entity recognition and disambiguation is already in use in several different applications. It is commonly used by search engines to disambiguate search queries, and deliver the best possible results. An example of this can be seen on Fig 1.1, where the query *michael jackson* has been entered into Google. A disambiguated guess of who is meant can be seen on the right. This is most likely who you think of when you hear the name *Michael Jackson*, although there are thousands of people out there with that exact name, and 21 of them famous enough to have their own Wikipedia page (Wikipedia, 2016b). Another linking can be seen on Fig 1.2, where the football player born 1973, with the same name is the disambiguated guess.

Entity recognition and linking is however usually a part of a system with a larger goal, such as automatic translation, topic detection or information retrieval. This is also the case in the example with *Michael Jackson*, as the goal is not disambiguation in itself, but to deliver correct search results.

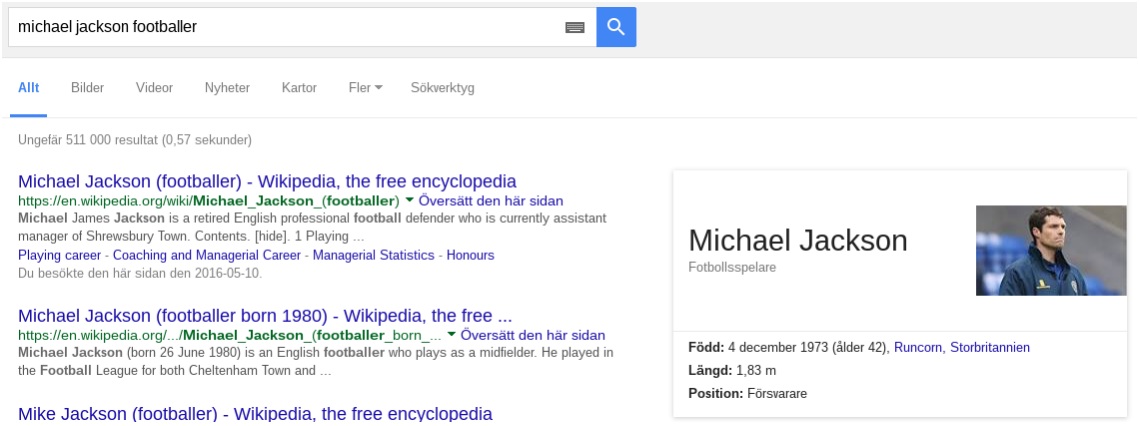

Figure 1.2: A different linked named entity, done through minimal context.

1.2.1 Intended Usage

The system described in this thesis is designed to be applied to large amounts of texts, using a cluster, in order to structure the text for usage of a larger system. This larger system, called Hajen, is meant to extract knowledge from text using additional tools in order to be able to answer factual questions in many different languages. This is a way of simplifying people's lives and lowering the boundaries to learn new things.

The system in this thesis is designed to assign unique identifiers to named entities, so that when the later steps in the Hajen pipeline tries to extract facts about a person or a place, it can be sure who or what the fact is about.

1.3 Goals & Constraints

This thesis aims to contribute to a solution to the problem of entity recognition and disambiguation with a focal point on named entity recognition, in several different languages. This means that within the problem of finding names in a text and giving them a unique identifier, the focus of this thesis lies in finding the names in the first place and assigning them possible candidates. Linking will also be explored, but not at the same depth.

The explored solutions are meant to work in a multilingual setting of 6 different languages, including Swedish, English, Spanish, French, Russian and German. This is due to that the system is to be a part of a research project at Lund University called *Hajen*, which is a multilingual question answering system under development.

Another constraint that has shaped the project is the time delay aspect of the system. The system should be able to be deployed on large corpora of text using a cluster. This means that the processing time of the system must be relatively small. It would also be preferable if the program is relatively self-contained, and lightweight in terms of memory.

1.4 Previous Work

A lot of work has been done in this field, within the last 10 years. The work of Bunescu and Pasca (2006) and Cucerzan (2007) both used Wikipedia as a knowledge source. They used the hyperlinks for names, the articles as entities, and semantic knowledge from redirect pages and list pages, in order to do disambiguation. Cucerzan (2007) however, deployed a system that did both entity recognition and disambiguation.

The work of Milne and Witten (2008) had similar ideas, using Wikipedia as a knowledge source for identifying and linking names, but also introducing a number of metrics for how to do so in a more efficient way, improving both recall and precision. The goal of their system was to find names in texts and link them to their corresponding Wikipedia article, calling the process *wikification*. This idea is the foundation of this thesis, although it has been altered slightly.

Ferragina and Scaiella (2010) introduced the problem of wikifying shorter pieces of text, and thus having less contextual clues to draw from. They claim to outperform Milne and Witten (2008), especially for shorter pieces of text, and do so using a number of tricks for utilizing the knowledge source of Wikipedia in better ways.

Another influential work is the CoNLL-2003 shared task (Tjong Kim Sang and De Meulder, 2003), that was about language independent named entity recognition, with a focus on German and English. The datasets provided in the shared task have been used, as well as the participants' results, as a form of comparison for how well the system described in this thesis is performing.

1.4.1 ERD'14

More recent work within the field was done in the ERD'14: Entity Recognition and Disambiguation Challenge (Carmel et al., 2014). The scope and evaluation method used in the challenge is what makes it interesting. They gave each competitor a piece of unlabelled text, and expected an output where each entity is recognized and disambiguated, meaning that a part of the text has been marked as a named entity, and assigned a unique ID, referring to that specific entity in a database. In the case of ERD'14, the database used was a subset of the now deprecated Freebase graph database (Bollacker et al., 2008). This approach contains the entire pipeline in an end-to-end fashion, more similar to that of a real world application.

The competition consisted of two different tracks. A long track, challenging the contestants with unlabeled documents with an average size of around 600 words, and a short track, where the unlabeled text was more along the size of a search query.

The method and dataset used for testing is also used in my work to evaluate the system that was developed. A few of the top scoring contributions to the challenge, namely the works of Lipczak et al. (2014), Eckhardt et al. (2014) and Piccinno and Ferragina (2014), have been influential for this thesis.

Participants

The contribution of Lipczak et al. (2014), named Tulip, was divided into two different parts: a spotter and a recognizer, where the spotter finds all potential mentions in a text

and the recognizer decides which mentions are worth keeping. Both parts are based on open source software, with the spotter being based on Solr Text Tagger (Smiley, 2013), and the recognizer being based on Sunflower, which was also built at Dalhousie University in parallel with Tulip. The authors state in the conclusion that the result relies more on the successful spotting and recognition of entity mentions, than it does on proper disambiguation. The team finished second in the long track of ERD'14, with an F1-score of 0.735, as well as being the system with the lowest latency, with an average processing time of 290 ms per document.

The WAT system (Piccinno and Ferragina, 2014), is based on the algorithms of TagME, which according to the authors, was the best system available. The system is, like Tulip, set up as a modular framework, with the system being split up into a spotter, a disambiguator and a pruner. This allowed the team to experiment with several different techniques for each part, with the main focus set on finding the best suited disambiguation algorithm, and tuning it to the task. According to the conclusions of the paper, this focus turned out to be a mistake as the difference between the best and worst algorithm was minimal. The team finished 8th of the long track in ERD'14, with an F1-score of 0.672, but with a precision of 0.876, which was the highest among all competitors.

The contribution to ERD'14 by Seznam Research (Eckhardt et al., 2014) used a fairly simple method of NER with a small knowledge base only consisting of the titles and redirects of Wikipedia pages, but extended each recognized mention into several smaller parts. For example if the system managed to tag *International Olympic Committee*, the system created a set of possible acronyms such as *IOC* and *I.O.C*, and divided the name into parts, *International*, *Olympic* and *Committee*, and tried to tag all of these mentions in the document with the same identifier as the original mention. Eckhardt et al. (2014) used a variation of the classic PageRank algorithm for disambiguation, which produced very stable results. They finished third in both the long track and the short track.

1.5 Contribution

This thesis contributes evaluations of multilingual methods of named entity recognition and disambiguation. It explores different methods and presents evaluations for how well the different methods work in English. It also presents a novel way, to the best of the author's knowledge, of pruning unsatisfying links in a collected knowledge base by clustering.

The system described in this thesis was primarily developed by myself, with a few exceptions.

- The weighted category graph described in section 3.5.2 was developed by Océane Chabrol and David Norrestam as a project in a course about natural language processing at Lund University.
- The collecting and counting of hyperlinks from Wikipedia was done with the help of Marcus Klang, a PhD Student at Lund University who also works with the Hajen research group.

Chapter 2
Method

The goal of this thesis is to structure text in a way that simplifies the process of understanding it for a machine. A part of doing so is recognizing and identifying names and concepts. If each entity in a sentence is marked with a unique identifier, the computer can analyze the verbs and adjectives and create logical statements, such as what the named entity has done, can do, or how it is described. By converting sentences into structured information in this manner, a machine can "make sense" out of natural language, and thus utilize the world's largest body of information: written text.

This chapter describes what steps are taken in order to achieve entity identification, how common problems are approached, and what the development process looks like.

2.1 Wikification

A way of identifying entities in text is called **wikification** as suggested by Mihalcea and Csomai (2007). The process utilizes the web based encyclopedia of Wikipedia and the links between the articles. Its goal is to learn which words are commonly linked and what address the link leads to, and then mark unlinked words with the best possible guess.

I start by gathering the links from a dump of Wikipedia, in similarity with Bunescu and Pasca (2006). Each link is seen as a **mention** of an entity and consists of the pair **surface form** and **entity**, where the label, or visible words, is called the surface form, and the address to which the link leads corresponds to the entity, as explained in Fig. 2.1. The algorithm collects each mention and counts the frequency of the entities and surface forms. This collection of surface form - entity pairs is called a **knowledge base**. The algorithm does this because each entity can have many different surface forms, or names, and the best source for those names is to see how the entity is actually referred to *in the wild*.

I use the knowledge base when confronted with unlinked text and **tag** each string in the document that matches a surface form in the knowledge base with all entities that are part of a mention with that surface form. This is because surface forms can have

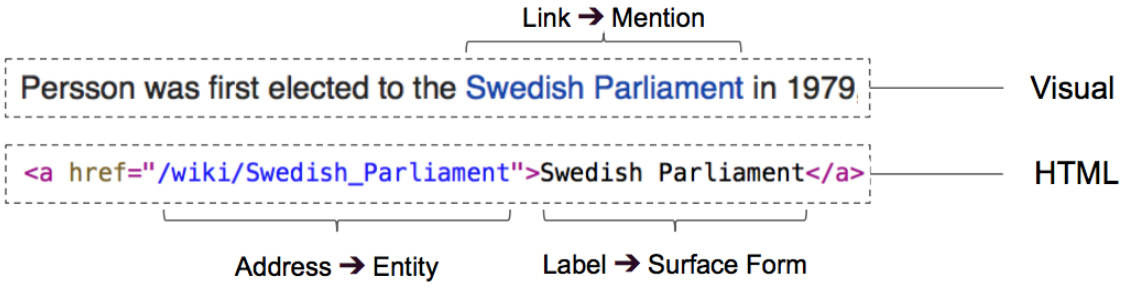

Figure 2.1: The structure of links in Wikipedia.

many different entities related to them. Borrowing the example of the mention *Texas* from Cucerzan (2007), we can see that it has many different meanings:

- *former Texas quarterback James Street*, should be linked to /wiki/University_of_Texas_at_Austin
- *in 2010, Lasse Stefanz released their album Texas*, should be linked to /wiki/Texas_(Lasse_Stefanz_album)
- *the characters in Texas include both real and fictional explorers*, should be linked to /wiki/Texas_(novel).

Given any of these sentences, the surface form *Texas* will be tagged with 86 possible entities. The process of deciding which entity is meant is called **disambiguation** or **linking**, and consists of taking one of the entities that the mention is tagged with and linking the mention to that entity's Wikipedia article. A very effective baseline is to select the entity that is the most frequent, given that surface form. I improve the linking with context analysis, through looking at what other entities are found in the text.

Another issue is to determine whether a piece of text actually refers to a name. There are many surface forms that consist of commonly used words or phrases, such as the Stephen King novel *It*, or the album by The Strokes called *Is This It*. The system uses a set of manually written rules that depend on such things as capitalization, dictionary information and punctuation. I also explore more complex approaches, like context analysis, in order to recognize named properly. This process is called **spotting** or **entity recognition**.

2.1.1 Wikidata

Wikidata is a free, linked database in the Wikimedia family that contains structured information about a large number of entities. It also connects the different language versions of each article in Wikipedia with a unique identifier called a Q-number. An example containing the entity /wiki/Riksdag can be seen in Fig 2.2.

I translate the entities from Wikipedia addresses to the Q-numbers used in Wikidata, in order to make the system less dependent on language. This way the system can reuse resources collected in one language in many different languages.

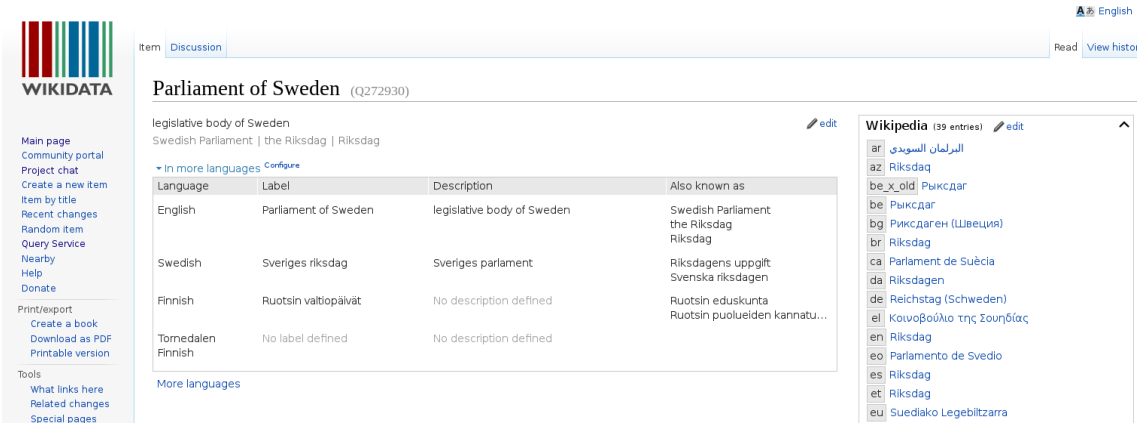

Figure 2.2: The Wikidata page of the Swedish Parliament, Q272930

2.2 Software Development Process

I began with developing a baseline, similar to the spotter in the system of Lipczak et al. (2014), using Solr Text Tagger and filters created by pre-processing. The development was done by reading the description of the system in the paper, and using similar or identical technologies. In the cases where the resources used in the article were not applicable to a multilingual context, I explored another way, or skipped the resource altogether. The baseline was then extended by taking suggestions from other articles as well as adding original ideas.

For every new idea that I implemented, or iteration of the algorithm, two different methods of evaluation were used, one statistical method and one visual method.

The statistical method was used to measure the performance of each iteration, and to only keep the changes that actually meant an improvement. This was done with the help of documents that have been annotated by humans. The method measured *recall*, which considers how many of the humanly annotated names the system can find, and *precision*, which measures how many of the systems annotations that are considered correct. These two values are then averaged into something called an *F1-score*. The exact definition of the measurements can be found in section 4.2.

The visual evaluation method was used to display the tagged output in an easy to understand manner, in order to visualize systematical errors of the algorithm.

Chapter 3
Implementation

The system is constructed as the pipeline visualized in Fig. 3.1, which consists of a database of names called a **knowledge base**, the open source software **Solr Text Tagger**, a set of constructed **filters**, an **extender**, and a **linker** to make the final decision. This chapter explains the different parts of the pipeline in detail.

In terms of a spotter and a disambiguator these systems traditionally get divided into, the entire system should be considered a spotter, since their job is to recognize names and to prune the unlikely, with an exception of the linker which acts like a basic disambiguator as well as a pruner.

3.1 Knowledge base

I construct the knowledge base from links on Wikipedia in line with the ideas of Mihalcea and Csomai (2007). As Wikipedia is a multilingual resource, with 282 active languages (Wikipedia, 2016a), its use does not restrict the project's intention of being available in the six different languages mentioned earlier.

Each link is seen as a **mention** which consists of a **surface form** and an **entity**. A surface form is the linked text the reader can see, and the entity is the address which the link leads to, as can be seen in Fig. 3.2.

Each mention is counted, and this count is referred to as **commonness** as suggested by Milne and Witten (2008), although it is not strictly identical. My metric is an integer sum, while they transform it into a ratio between 0 and 1 calculated as the following, given that S is a surface form and E is an entity:

$$P(E|S) = \frac{\sum \text{S leads to E}}{\sum S}$$

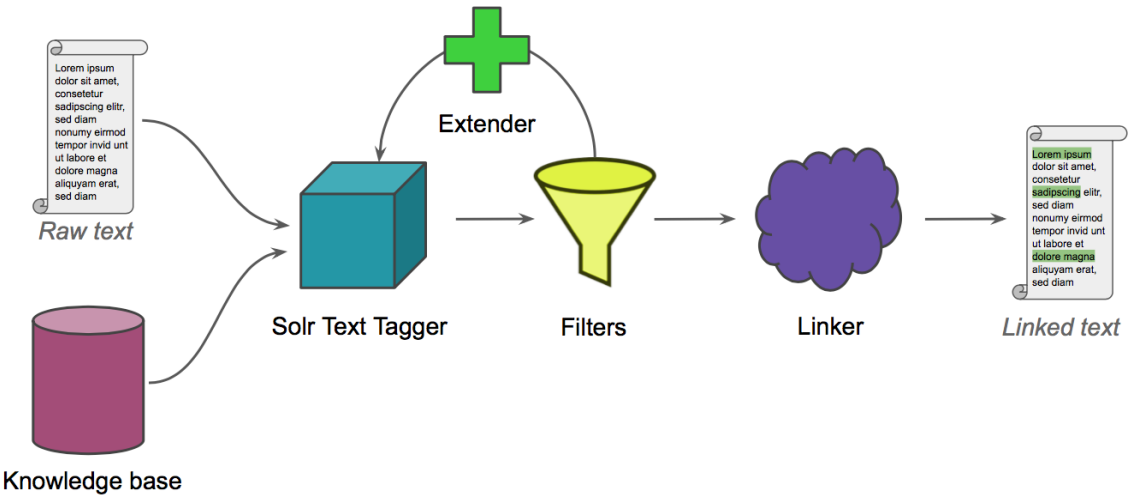

Figure 3.1: The pipeline of the system

3.1.1 Entity Translation

In order to use a language agnostic identifier, I translate the Wikipedia address, or Wikilink, used as the identifier for an entity into a Wikidata Q-number. The translator does this by downloading a dump of Wikidata, and for each item in the dump extract the Q-number and the name of the article for each language. Since the dump does not contain any URL, the algorithm must recreate the address by taking the title and replacing spaces with underscores. This creates a translation table that has around 90% in coverage.

The 10% that are not covered by the translation are due to the fact that Wikipedia has something called *redirect pages*, functioning like an alternative URL for some entities that have an alternative name, or to cover common misspellings. The algorithm uses the part of the Wikipedia dump that shows which Wikipedia pages redirect to other Wikipedia pages, as well as a translation table between Wikipedia IDs and Wikilinks, which improves the coverage rate to 99.6%.

The entity translation has two effects:

1. Unifying the identifier across languages, allowing for the reuse of resources, such as link information.

2. Unifying different identifiers that actually refer to the same entity.

3.1.2 Pre-processing

Since a part of the goal of this system is to have low latency, I do as much offline pre-processing to the knowledge base as possible, in order to keep down the latency of the system. The pre-processor creates a number of possible flags for each mention that are to arouse suspicion during the later steps in the pipeline. The following flags are created:

1. I use a dictionary of common nouns, verb and adjectives for each language. If a surface form only consists of words in the dictionary, the system marks the mention

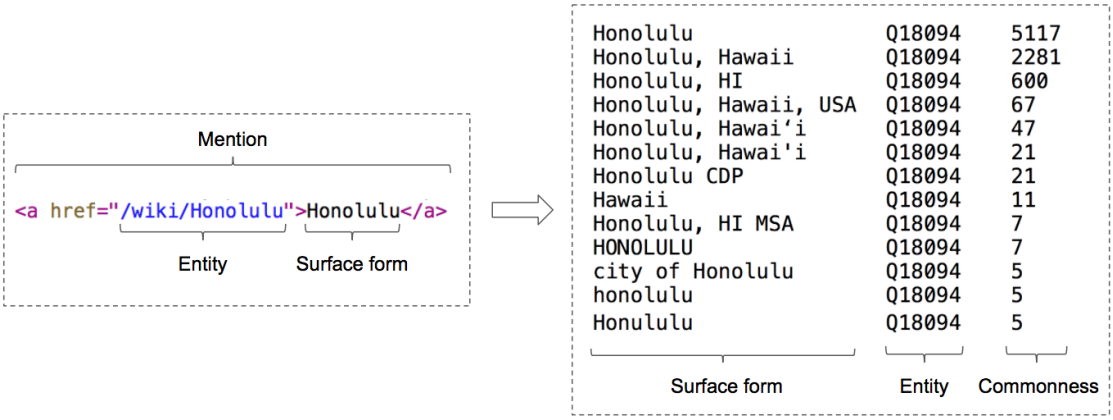

Figure 3.2: An example of the surface form-entity relationship, using the entity Q18094 most commonly called Honolulu.

with the flag `only-dictionary`. An example of this would be the artist *Prince*, that can also refer to the concept of royalty.

2. I collect a dictionary of **stop words** from Wikipedia for each language. Stop words are the most frequent words in a language. The English stop-word dictionary contains words such as *the*, *in*, *and* and *a*. If the all the words in a surface form are in the stop word dictionary the mention is given the flag `only-stop-words`.

3. The system marks mentions that have a high number of entities linked to them with the flag `common`. An example of this is the surface form *John* or the surface form *details* with almost 5000 entities linked to each.

4. I mark all mentions that lack upper-case letters with the flag `lower-case`. This is done using a simple regular expression that searches for uppercase letters.

5. The system treats personal names with a special interest. If the most common surface form of a person has several words, each surface form only consisting of one word is given the flag `generic`. An example of this would be the surface form *Bush*, referring to the former president George W. Bush. The reason for this being that there are many people with the last name Bush, with only a very small portion of them having their own Wikipedia article.

6. The algorithm calculates the total sum of the commonness for all the different surface forms of an entity and call it `total-commonness`.

3.1.3 Pruning

Although Wikipedia is peer reviewed, and fairly well written, there is plenty of noise in the collected dataset. Take this sentence as an example:

To read more about Ludwig XVI click here.

If the phrase *click here* is the part that is linked, the knowledge base will contain that phrase as a possible name for Ludwig XVI. This needs to be filtered out without affecting the well written links.

I determine the effectiveness of each filter by implementing it, and displaying the candidates that would be filtered out when it is used. The ratio between poorly linked entities, and well linked entities is manually counted and calculated for a randomly selected subset. If the filter is 90% accurate or more, it is considered successful.

The system contains the following filters:

1. The mention is marked as `lower-case` and either `only-dictionary` or `only-stop-words`

2. The `commonness` of the mention is exactly 1, while the `total-commonness` of that entity is sufficiently high.

3. The mention consists of several words, and starts with a lower-case stop word, that can not be considered a definite article. This is highly effective.

One of the more interesting pruning methods I experimented with is looking at all the surface forms for each entity, and calculating a modified Levenshtein distance (Levenshtein, 1966), also known as edit-distance, between them. The edit-distance is modified to be a ratio of the longest surface form, instead of an integer distance. If two surface forms are sufficiently close, they are grouped together. When all surface forms have been considered, the mentions without any group are considered outliers and pruned.

This method works well for entities with many different surface forms, so that the alternative names for an entity have different casings, or exist in both normal and possessive form. It works less well for less common entities with an alias that only has one version of the surface form.

3.1.4 Filtering to fit test data

The test set that was supplied in the ERD'14 competition for development came with a list of slightly more than 2 million entities, which the system should be able to recognize. This way the issue of defining a name was avoided. Since the English version of Wikipedia contains many more articles than that, I filter the knowledge base to fit this specific need. This subset of the knowledge base is treated like a specific language, and all evaluation done against ERD's development set is queried against the knowledge base of this "language".

3.2 Solr Text Tagger

Solr Text Tagger (Smiley, 2013) is a piece of open-source software by David Smiley, developed for OpenSextant, a U.S. Government software project for geospatial and temporal extraction capability. Solr Text Tagger's goal is to be a fast and memory efficient text

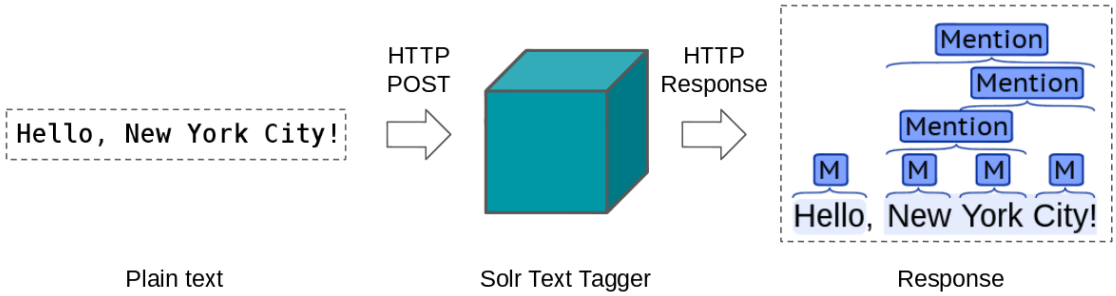

Figure 3.3: A visual representation of the communication with Solr Text Tagger

tagger. Given a database of names, and a piece of untagged text, it will mark all the occurrences of all names in the database, and since it is based on the search platform **Apache Solr**, which in turn is based on the full-text indexer **Apache Lucene**, it can utilize features such as flexible and phonetic matching.

The memory efficiency and high speed performance of the software mainly comes from the use of finite state transducers, implemented in Lucene, and that the choice for string searching fell on the Aho-Corasick algorithm, using a single pass to find all names in a dictionary (Aho and Corasick, 1975).

Solr is a standalone HTTP server, with a REST API. Once the knowledge base is sent to the Solr server and indexed by Lucene, a simple POST request, containing the text one wishes to have tagged, will result in a JSON response with the full text and all possible tags Solr Text Tagger has found. A visual representation of the process can be seen in Fig 3.3

3.2.1 Finite State Transducers

An FST is a finite state machine, but with both an input and an output. This can be used to effectively translate between two different formal languages. Given an input as a string of characters, a finite state transducer can produce an output that is a different string of characters. In Fig 3.4, an input string of `abccd` would produce the output string `hello`.

The idea is to have the input labels correspond to the letters and symbols of mentions and the output labels to form identifiers for different entities, and fetch the data around those entities from a separate database. This can be very memory efficient and fast.

3.3 Filters

The output of Solr Text Tagger is very noisy, with almost every word considered a match of some mention in the knowledge base. The result has very high recall, but very low precision. I apply a filter that attempts to only allow the correctly tagged mentions in order to improve the precision without affecting the recall.

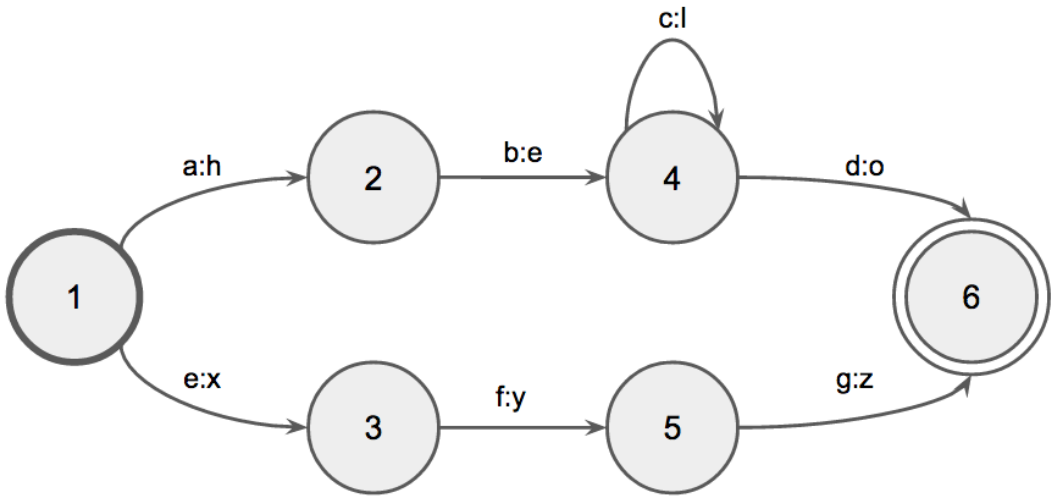

Figure 3.4: An example of a finite state transducer

3.3.1 Manually Written Rules

The main part of the filters are written as rules based on syntactical clues and the flags set in pre-processing. Each rule is evaluated against two different development sets, and only kept if it results in an improvement.

The most obvious, and most effective, filter is removing any surface form that does not contain a capital letter. This loses some recall, but increases the precision dramatically. I improve this by creating a function called `looksLikeAName()` that takes the number of capital letters, whether the mention is the start of a new sentence and whether the mention has the `only-dictionary` tag into consideration.

Since all mentions with capital letters are allowed, this creates issues for the parts of text that consist of capitalized words, such as headlines. The system uses a function based on regular expressions to recognize headlines. Mentions in headlines with the `only-dictionary`, or `only-stop-words` tag, are marked as suspicious.

Mentions with the tag `generic` are removed in the first pass, and stored in a different list for re-evaluation. Once the document has been tagged, the generic names are reconsidered. If a mention of the same entity that is not generic is already tagged in the text, the generic name is retagged in the text. The idea is that a mention such as *Bush* is only used when there is a full mention of *George W. Bush* somewhere else in the text.

I do this to avoid tagging people that are not on Wikipedia. The name *Jenny Bush* for example, does not exist in the system's knowledge base, but both the name Jenny and the name Bush exist as generic names, and the system would thus generate two separate mentions, that would both be incorrect.

Although the system contains a function to avoid multiple tagging of personal names, the knowledge base still contains mentions that are not personal names, but can easily be confused with them. One example is the town Obama in Fukui, Japan. This can easily create a double tagging of a name that is not present in the knowledge base. Two different systems are developed to avoid this. The first system uses regular expressions to conclude if the mention is part of a series of capitalized words. This is called `Con-`

`tainedByRegexName` and is used to raise suspicion. The other system looked at the start and end of tags. If two tags are immediate neighbours, with the exception of a space, they are removed. The system considers all the candidate entities for a surface form, and only prune the mention if at least one of the candidate entities is a generic name.

Other filters that were used to arouse suspicion were:

1. The `only-stop-words` tag.
2. The `only-dictionary` tag in combination with that the number of words in the surface form equalled to 1.
3. A function called `isDefaultSense()`, which uses `commmonness` and `total-commonness`, to determine if the mention in question is the default way to refer to a specific entity.
4. If the commonness of a mention is sufficiently low, more suspicion is aroused.

The implementation of the function, with all the manual rules, can be found in Appendix A.

3.3.2 Logistic Regression

I also conducted an experiment with logistic regression, using the AIDA training dataset introduced in the CoNLL 2003 shared task (Tjong Kim Sang and De Meulder, 2003): a humanly annotated, non-noisy dataset in English, consisting of more than 23 000 mentions.

The dataset is tagged with the system, but without any filtering mechanism. Each entity for each surface form is labeled as either correct (1) or incorrect (0), depending on if the mention overlapped a humanly annotated mention.

Logistic regression is a mathematical model, developed in the late 1950s, to model and predict the outcome for one categorical dependant variable y, given several real independent explanatory variables $x_1, x_2, x_3 ... x_n$.

In a simplified way it can be described as finding the β_i parameters that best fit

$$\begin{cases} y = 1, \text{ if } \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_n x_n + \varepsilon > 0 \\ y = 0, \text{ otherwise} \end{cases}$$

where ε is an error distributed by the standard logistic distribution, and the β parameters are estimated with an algorithm called **maximum likelihood estimation** (MLE).

The features that are also used for the manual rules are used as explanatory variables, as well as a set of combinations of features, that are dependent. These combinations are found using a decision tree, that I generate with the machine learning kit Weka (Hall et al., 2009), and expressed as:

$$FeatureC = FeatureA \times FeatureB$$

All features are converted from Boolean to either 0 or 1, and trained using liblinear-java, a Java port of the liblinear library (Fan et al., 2008).

The system balances the trained dataset to contain as many positive as negative examples, by cloning the set of examples that are in minority, until the set is balanced. This is done to avoid bias in the prediction.

3.3.3 Mention Probability

I calculate a metric called **mention probability** as suggested by Eckhardt et al. (2014). It is an attempt to interpolate which surface forms are mostly used as a name of an entity, and which are most commonly used as words.

The unfiltered system tags all of Wikipedia and creates a ratio from the number of times a surface form (SF) exactly match a Wikipedia link, to the number of times that are attempted for that surface form.

$$\text{Mention probability} = \frac{link(SF)}{freq(SF)}$$

The surface form *Medical Center*, for example, is linked 1.0% of the times it is used, while the surface form *Medical Center of Central Georgia* measures 73.7%. This can be used to determine what to prune in a language agnostic manner.

3.3.4 Clean-up

None of the datasets that are used for evaluation considers overlapping mentions, and thus every overlapping mention that is kept for evaluation will result in at least one incorrectly marked mention. In order to determine which mention to keep in case of an overlap, a fairly simple algorithm is used, called **longest dominant right**. The idea is to keep the mention that is longer. If they are both of the same length, the mention that is further to the right is used. Given the sentence:

Priority mail costs are calculated by the U.S. Postal Service.

Both *U.S. Postal Service* and *Postal Service* will be marked as mentions. Only the longer mention will be kept by the algorithm.

In an early version of the system, this was set as an option for Solr Text Tagger, as suggested by Lipczak et al. (2014). This would leave the system with long but incorrect tags that were correctly discarded, such as *in Afghanistan*, and no sub tag *Afghanistan*, to take over the role of the longest dominant right.

3.4 Extender

I created an extender to improve the recall of the system, as suggested by Eckhardt et al. (2014). Once a mention has been tagged, and not been filtered out, it is considered for extension. For each mention consisting of several words, the extender creates new surface forms from that mention both by creating acronyms and by splitting it into multiple parts.

Given the text *...a division of First Citizens BancShares Inc. of Raleigh, N.C.*, the system is able to recognize the surface form *First Citizens BancShares Inc*. The extender

creates possible acronyms for this surface form, such as *FCBI* and *F.C.B.I.* It also looks for parentheses, immediately following a mention, giving a suggestion of how the surface form is meant to be abbreviated.

The extender then splits the surface form into parts of 1, 2 and 3 words. The mention above would generate *First*, *Citizens*, *BankShares* and *Inc.*, as well as *First Citizens*, *Citizens BankShares*, *BankShares Inc*, and so forth. This is an improvement compared to the system of Eckhardt et al. (2014), in which each surface form is only split into parts of one word.

All the generated parts are given each possible candidate the original mention has, and the system then attempts to find the extended parts in the same text as where the original mention is found. The tagged extensions are filtered in the same manner as all other tags. When the example text above continues:

First Citizens also owns IronStone parent Atlantic States Bank, which has 34 offices in Florida and Georgia.

The system is able to recognize *First Citizens*, and correctly tag it with the same candidates as before, without that specific surface form ever being seen in the original dataset collected from Wikipedia.

3.5 Linker

The linker has the job of selecting which entity is meant by each name. It does this by using PageRank to prune the unlikely candidates, and commonness to select the most likely of the ones that remain. The weighted category graph that is also described in this section tries to widen the named entity recognition by using the candidates that are already linked.

3.5.1 PageRank

I improve the candidate selection and mention pruning by applying a modified version of PageRank to the tagged mentions, in a style similar to Eckhardt et al. (2014). PageRank (Brin and Page, 1998) is an algorithm originally developed for ranking web pages, by Larry Page. It considers the pages of the web as a graph of nodes, and the links between the pages as directed links in the graph.

Each node in the graph is given an initial value of $1/N$, where N is the total number of nodes in the graph. The rank of each page is then determined by the links leading to that page, the number of links leading away from the page, and a damping factor d between 0 and 1. For any page u, the PageRank value is given by the following formula:

$$PR(u) = \frac{1-d}{N} + d \times \sum_{p_j \in M(u)} \frac{PR(p_j)}{L(p_j)}$$

Where p_j is any page in the graph, $M(u)$ is the set of pages that have links to u, and $L(p_j)$, is the number of links leading away from page p_j.

This formula is iterated over the graph several times, until the change between each iteration is sufficiently small. Since the sum of all values in the graph always equals to

1, the values converge, and the difference in value for each node between each iteration declines.

I create a node for every surface form-entity pair that is detected in the text, and run a few iterations of PageRank. The linker does this to determine which entities usually appear together, and prioritize these over entities that have not been seen in the same context earlier.

To determine which entities are linked to each other, the internal links of Wikipedia are analyzed. Two entities are considered linked if:

1. The article of Entity A links to the article of Entity B.

2. A link to both the article of Entity A and the article of Entity B occurrs in the same paragraph.

The system has a cutoff value for how many links between two entities are needed for them to be considered linked. Any two entities with fewer connections than the cutoff value are pruned. This reduces the memory load, and only keeps the links that are sufficiently common. All links are two-way links, as it is hard to determine which entity links to which in the case of two links occurring in the same paragraph.

Unlike the work of Eckhardt et al. (2014), I initialize each node with the standard value $1/N$, and the final value is divided by the initial value, and used to affect the suspicion for a candidate.

A second round of filtering is applied after the PageRank algorithm has concluded, and the candidates with a suspicion value that is too high are cut.

3.5.2 Weighted Category Graph

I use a category graph of the Wikipedia articles, with calculated weights for each category along the ideas of Sunflower, as mentioned in Lipczak et al. (2014). This is done with the help of the user-annotated categories in all the languages that exist for each article. The more languages a category appears in, the higher weight it receives. If an article has the same category in all languages, that category receives the maximum weight of 1.0.

This is done to differentiate the important categories from the less important ones. Take the English Wikipedia article on George W. Bush, as an example. That article is marked with 74 different categories, the date of writing, with everything from *Presidents of the United States* to *Painters from Connecticut*. Both these categories are correct, but one of them is a bit on the obscure side. An example output of the system can be seen in Table 3.1.

The weighted category graph is used in the following way:

1. The filter is set to output a high-precision, low-recall set of entities. This means that they are the entities the system is the most sure are correct. These entities have their category-vector calculated, and a **topic centroid** is formed as the linear combination of these vectors.

2. The topic centroid, and the entities it consists of, is put into question. This is done by comparing each of the high-precision candidates with the topic centroid with a

Category	Ratio
Sweden	1.00
Countries in Europe	0.38
Member states of the European Union	0.30
Constitutional monarchies	0.20
Scandinavia	0.20

Table 3.1: Top 5 categories for Sweden, Q34

cosine similarity. If they score under a certain threshold value they are pruned off. Finally, the topic centroid is recalculated.

3. Each candidate in the unfiltered tagged output, is then compared to the topic centroid with a cosine similarity. The entities that have above a certain threshold value are considered good, and added to the final set of candidates.

3.5.3 Entity selection

Since each surface form has several candidates as to which entity is meant, the linker needs to do some sort of selection or disambiguation. The system uses a very simple baseline by sorting the entities by commonness and selecting the entity with the highest value.

The baseline of commonness is quite hard to beat according to Ferragina and Scaiella (2010), and has a correctness of at least 80% according to my own experiments with the system and the ERD-51 dataset that is explained in detail in section 4.1.

3.6 Visualizer

I visualized the output of the system in order to allow for rapid exploration of the data. The visualizer is implemented as a website, with a text-field where the user can write or paste text, a possibility to select language, and a submit button. An example response from the demo can be seen in Fig 3.5.

The text gets annotated with clearly visible labels. Once the user hovers over the label, the different candidates for each mention are displayed in the order of disambiguation, with a link to the website of the Wikidata page for that entity.

This allowed me to quickly find systematical errors of the system during the development process, as well as enabling me to quickly share the concept and issues of my work with others, and thus creating meaningful discussions around the subject.

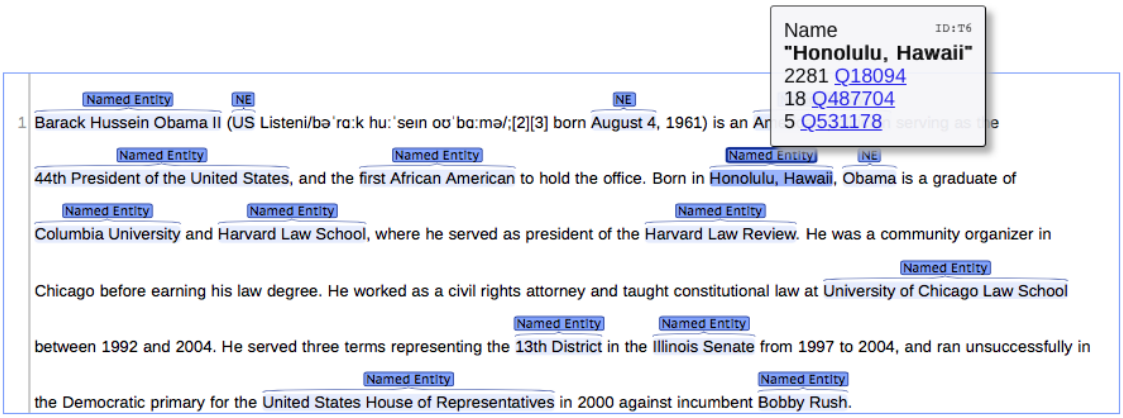

Figure 3.5: Visualizer

Chapter 4

Evaluation

I continuously evaluated the system with a statistical method over several different datasets, on both named entity recognition as well as the full pipeline of entity recognition and disambiguation. The results are presented as a series of different iterations that the system went through for each dataset and evaluation type.

4.1 Datasets

The system was evaluated with two different datasets for English, one is used for both named entity recognition (NER) and disambiguation, while the other is only used for NER. I did not have access to evaluation sets for the other languages.

ERD-51: The evaluation uses the development dataset from Carmel et al. (2014) for named entity recognition and disambiguation. It consists of 51 documents that have been scraped from a variety of sources on the Internet, with 1169 humanly annotated mentions, where each annotation has a start, an ending, and a Freebase identifier (Bollacker et al., 2008) linking each entity. In the competition, a set of entities, slightly over 2 million, was given, and thus the problem of defining what a named entity actually is was avoided. The systems entity database is filtered to only contain mentions of entities from the given set.

AIDA/YAGO: I evaluated named entity recognition in English with the AIDA/YAGO dataset, collected from the CoNLL-2003 shared task (Tjong Kim Sang and De Meulder, 2003) of named entity recognition. It is a collection of 447 news articles with 11491 humanly annotated names. The dataset groups the names into 4 different categories: persons, organizations, locations and miscellaneous. Since the dataset is for NER, it lacks identifiers for each entity, and thus can not be used for disambiguation.

4.2 Evaluation metrics

The evaluation metric I used for disambiguation is identical to the metric used in Carmel et al. (2014). Given a document with a set of humanly annotated linked mentions L_j, with a starting character offset (s_j), an ending character offset (t_j) and a linked entity id (e_j): $L_j = (s_j, t_j, e_j)$ as well as a set of annotations created by the system $\hat{L}_j = (\hat{s}_j, \hat{t}_j, \hat{e}_j)$. An annotation created by the system is considered a match if:

$$Match(\hat{L}_j, L_j) = \begin{cases} \hat{e}_j = e_j, \text{ and} \\ (\hat{s}_j, \hat{t}_j) \text{ overlaps with } (s_j, t_j) \end{cases}$$

The method for evaluating the named entity recognition is very similar, except for the removal of an entity identifier from each mention: $L_j = (s_j, t_j)$. An annotation is considered a match if:

$$Match(\hat{L}_j, L_j) = (\hat{s}_j, \hat{t}_j) \text{ overlaps with } (s_j, t_j)$$

Both evaluation methods have the same final evaluation metric. Let B_i represent the set of humanly annotated links, and $\hat{B}_i$ represent the output of the system, for each document. $Match(\hat{B}_i, B_i)$ is the sum of positive $Match(\hat{L}_j, L_j)$ for each document. Precision and recall are calculated and a micro-averaged F-measure is defined as the harmonic mean of the two:

$$Precision = \sum_i Match(\hat{B}_i, B_i) \,/\, \sum_i \hat{B}_i$$

$$Recall = \sum_i Match(\hat{B}_i, B_i) \,/\, \sum_i B_i$$

$$F\text{-}measure = \frac{2(Precision \times Recall)}{Precision + Recall}$$

Given that named entity recognition is a subtask of ERD, the results of NER will always be more generous.

4.3 Results

The tables in this section present the results in chronological order, and thus the performance improvement over time can be easily understood. The tables also show the performance of different implementations, allowing the reader to extract how each part of the pipeline affected the result. The execution time is normalized to time spent per 5000 characters, on my machine. It should not be used as an absolute value, but can be used to relatively compare the different algorithms.

4.3.1 AIDA/YAGO dataset

Named Entity Recognition

Algorithm	Execution (ms)	Precision	Recall	F-measure
Baseline	-	0.940	0.630	0.755
With extender	-	0.889	0.684	0.773
Removed neighbour nodes	-	0.923	0.678	0.782
Simplified NER rules	-	**0.934**	0.705	0.803
Simplified NER rules, relaxed	-	0.912	0.769	0.835
Filtered out candidates with low commonness	1088	0.888	0.789	0.835
Improved extender	1157	0.905	**0.860**	0.882
Simplified the search, only using case sensitive string matching	717	0.932	0.851	**0.890**
Logistic regression	**598**	0.906	0.854	0.879

4.3.2 ERD-51 dataset

Named Entity Recognition

Algorithm	Execution (ms)	Precision	Recall	F-measure
Baseline	-	0.745	0.686	0.714
With extender	-	0.673	0.762	0.715
Remove neighbour mentions	-	0.698	0.730	0.714
Simplified NER rules	-	0.723	0.720	0.721
Simplified NER rules, relaxed	-	0.642	**0.859**	0.734
Is part of title	-	0.691	0.796	0.740
Filter out candidates with low commonness	250	0.718	0.785	0.750
ContainedByRegexName	165	0.788	0.764	0.776
Logistic Regression	**145**	0.544	0.857	0.665
PageRank	246	**0.865**	0.765	**0.812**

Disambiguation

Algorithm	Execution (ms)	Precision	Recall	F-measure
Baseline	-	0.551	0.521	0.535
With extender	-	0.520	0.581	0.549
Removed neighbour mentions	-	0.536	0.572	0.554
Simplified NER rules	-	0.613	0.610	0.611
Simplified NER rules, relaxed	-	0.522	**0.720**	0.605
Is part of title	-	0.593	0.684	0.635
Filter out candidates with low commonness	250	0.612	0.672	0.640
ContainedByRegexName	165	0.668	0.647	0.657
Logistic Regression	**145**	0.436	0.688	0.534
PageRank	246	**0.747**	0.661	**0.701**

4.4 Discussion

Given the results on the ERD-51 dataset, and the results that the competitors of ERD'14 competition had, the project is neither a success nor a failure. The results of HERD would have placed it on a 6th place on the long track, out of the 17 competing systems. It is about 6 percentage points lower than the highest scoring participant (Cucerzan, 2014), around 3 percentage points lower than Tulip, and 2 percentage points from the work of Eckhardt et al. (2014).

The difference in score between HERD and top contestants, mainly lies in the time restrictions for this thesis. If more time was spent integrating the different algorithms and metrics, it is my firm belief that the results would get closer to the results of Lipczak et al. (2014) and Eckhardt et al. (2014).

Part of the reason for the lower score is the multilingual nature of the project. It was sometimes not possible, or wise to use some of the resources or methods described in the competing systems, due to the fact that it was not available in other languages, and thus didn't fit the scope of the project. An example of this would be Google's Wikilink Corpus, a collection of links similar to the internal links of Wikipedia, that is only available in English.

Another reason for the difference in score is that there was as a lack of focus on creating the baseline. A bit too much time was spent exploring different techniques and options, such as logistic regression and knowledge base pruning. One can, on the other hand, argue that this is a learning experience and the exploration is necessary for the system to be developed in the best possible way, and for the thesis to have merit.

Comparing HERD's result on the AIDA/YAGO dataset to the competitors of CoNLL-2003, the system performs very well, with a higher F1-score than the highest of the competitors. It should be said, that the evaluation metric differs between CoNLL-2003 and ERD'14, and HERD is only evaluated with the ERD'14 metric. Thus the scores are not strictly comparable.

The reason for the difference in score between the ERD-51 and the AIDA/YAGO dataset mainly lies in the difference between the datasets. AIDA/YAGO is collected from well written news wires, and is mainly written by professionals with proper punctuation and capitalization, while ERD-51 is a collection of more poorly written texts collected from a wider variety of resources, spanning from blogs to eBay sales sites.

4.4.1 Multilingual Performance

It would have been preferable to evaluate the system on all languages it is being developed for, as it is otherwise not really possible to say anything at all about the multilingual properties of the system. The system has however been manually, and somewhat unscientifically evaluated with the visualizer for Swedish and French, by persons with domain knowledge, and the quality of the output appears to be comparable with the English system.

Manually Written Rules

The usage of manually written rules is suboptimal, as the rules will change for every language, which reduces the need for a corpus in exchange for domain knowledge of the language and time to be spent on developing rules. This is not an optimal design for a multilingual system.

I could, on the other hand, argue that a machine learning system will not only need an annotated corpus for a language, but also an analysis of which features are relevant, which in turn also requires domain knowledge and time.

A method that only uses Wikipedia as its corpus and avoids syntactic clues altogether, would be the best long-term solution.

4.4.2 Latency

With a latency of around 200ms per 5000 characters, the system is estimated to be able to tag the entirety of the English Wikipedia, using the cluster available at the computer science department in Lund University, in under a day. This calculation is done using the estimations that the average length of a Wikipedia article being around 2000 characters, that there are 5 million articles, 24 available cores and 100% overhead time.

The difference in latency between the different datasets are due to the fact that the articles in the AIDA/YAGO datasets are much smaller, and thus require a larger overhead time, as there is more communication with Solr Text Tagger and parsing of results per word.

The latency of the system could be optimized further with a few different methods. If one were to reimplement the features of Solr Text Tagger directly in Java, and thus avoid the HTTP-calls for a local web server as well as the parsing of the JSON that is returned as a result, the system could cut its execution time in half.

4.4.3 Logistic Regression

The logistic regression does not work as well as the manually written rules, especially for the ERD-51 dataset. The reason for the inefficiency could be the manner in which

it is implemented. Since every entity is marked as correct for a mention that matches a humanly-annotated mention, even if it is not the correct entity, it creates a bit of noise that makes it more difficult for the machine learning algorithm. This is because I lack a linked data set that is large enough to train on, so it is not possible to determine which candidate entity is correct for a mention.

The reason for the explorations done with logistic regression is the problematic nature of manually written rules, as these are not only language specific, but also requires domain knowledge of the language from the person composing the rules. This can however be avoided by using decision trees as was done while developing the logistic regression. Decision trees are somewhat slow when implemented, but are very similar to, and can easily be rewritten into, rules. Thus the requirement of domain knowledge can be transferred into a requirement of an annotated dataset for each language.

Chapter 5
Conclusions

This thesis explores different methods for high speed, language-independent entity recognition and disambiguation, with a strong focus on the recognition part of the process. The version of the system that had the highest score used a 4-step pipeline, ending with a light, personalized version of PageRank and a few manually-written, somewhat language agnostic rules. The final system had a weighted F1-score of 0.701 on the ERD-51 dataset.

The paper also explores the use of logistic regression for named entity recognition, as well as a weighted category graph, in order to determine which entities belong together.

The final system can be deployed to a large corpus such as Wikipedia, in order to uniquely identify concepts and names with a reasonably high recall and precision. Thus it will be possible to use in the later steps of the research project it is a part of, namely Hajen.

5.1 Improvements

There are a number of ways to improve the system, where the most interesting one is a combination of PageRank, that is precision improving, and the weighted category graph, that raises the recall. In an iteration of these algorithms a possible performance improvement can be seen.

An interesting metric to use in the improvement process would be **relatedness** (Milne and Witten, 2008), which was commonly used in disambiguation systems a few years ago, and is still used in a lot of systems today, among those a few of the systems in ERD'14.

Another interesting metric that I calculated for English during this project, but never used in any meaningful way, is a metric called **mention probability** as suggested by Eckhardt et al. (2014). This can be interesting data for the filtering part of the pipeline, and is a possible step to move away from manually written rules.

The system is implemented in three languages, namely Swedish, English and French, but can easily be extended to cover Russian, German and Spanish, as a pipeline of data

collection, pruning, tagging, filtering and disambiguation is already in place. It is a matter of downloading the Wikipedia dump, and letting the pipeline run through the different steps.

It would also be interesting to look at older systems, like the winning machine learning algorithms that competed in the CoNLL-2003 shared task, which mainly focused on character order and capitalization. The risk of these algorithms is that they require large amount of training data in order to produce decent results. This is not really available in all languages, and thus becomes a language specific method.

Another classic approach, that could improve the overall results, is to use the metric **term frequency-inverse document frequency** (TD-IDF) of the immediate surroundings of each mention, and try to find the words that commonly occur in the vicinity of an entity.

I am also tempted to explore the path of neural networks, that have become increasingly popular lately due to the success of a number of projects from Google, as well as the availability of open-source software such as TensorFlow that simplifies the process. The downside of neural networks is that they demand even more data than traditional machine learning techniques, and thus it becomes even harder to replicate the results for different languages.

The most drastic improvement would be to merge the output of the system with the output of another system, to form an ensemble system, or a stacked system. As calculated by Carmel et al. (2014), the top three systems of ERD'14 would have a recall of around 90% if they were to be combined. This requires some additional training and decision on what output to keep, in order to avoid a steep precision decline. This is a way to avoid implementing each of these improvements, and try to keep each system fairly simple, to later integrate the results.

Bibliography

Aho, A. V. and Corasick, M. J. (1975). Efficient string matching: An aid to bibliographic search. *Commununications of the ACM*, 18(6):333–340.

Bollacker, K., Evans, C., Paritosh, P., Sturge, T., and Taylor, J. (2008). Freebase: A collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, pages 1247–1250, New York, NY, USA. ACM.

Brin, S. and Page, L. (1998). The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, 30(1-7):107–117.

Bunescu, R. C. and Pasca, M. (2006). Using encyclopedic knowledge for named entity disambiguation. In *European Chapter of the Association for Computational Linguistics*, volume 6, pages 9–16.

Carmel, D., Chang, M.-W., Gabrilovich, E., Hsu, B.-J. P., and Wang, K. (2014). ERD'14: Entity recognition and disambiguation challenge. In *ACM SIGIR Forum*, volume 48, pages 63–77. ACM.

Cucerzan, S. (2007). Large-scale named entity disambiguation based on wikipedia data. In *Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, volume 7, pages 708–716.

Cucerzan, S. (2014). Name Entities Made Obvious: The Participation in the ERD 2014 Evaluation. In *Proceedings of the First International Workshop on Entity Recognition & Disambiguation*, ERD '14, pages 95–100, New York, NY, USA. ACM.

Eckhardt, A., Hreško, J., Procházka, J., and Smri;, O. (2014). Entity linking based on the co-occurrence graph and entity probability. In *Proceedings of the First International Workshop on Entity Recognition & Disambiguation*, ERD '14, pages 37–44, New York, NY, USA. ACM.

Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R., and Lin, C.-J. (2008). Liblinear: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874.

Ferragina, P. and Scaiella, U. (2010). Tagme: On-the-fly annotation of short text fragments (by wikipedia entities). In *Proceedings of the 19th ACM International Conference on Information and Knowledge Management*, CIKM '10, pages 1625–1628, New York, NY, USA. ACM.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. (2009). The weka data mining software: An update. *SIGKDD Explorations Newsletter*, 11(1):10–18.

Levenshtein, V. I. (1966). Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, 10:707.

Lipczak, M., Koushkestani, A., and Milios, E. (2014). Tulip: Lightweight entity recognition and disambiguation using wikipedia-based topic centroids. In *Proceedings of the First International Workshop on Entity Recognition & Disambiguation*, ERD '14, pages 31–36, New York, NY, USA. ACM.

Mihalcea, R. and Csomai, A. (2007). Wikify!: Linking documents to encyclopedic knowledge. In *Proceedings of the Sixteenth ACM Conference on Conference on Information and Knowledge Management*, CIKM '07, pages 233–242, New York, NY, USA. ACM.

Milne, D. and Witten, I. H. (2008). Learning to link with wikipedia. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management*, CIKM '08, pages 509–518, New York, NY, USA. ACM.

Piccinno, F. and Ferragina, P. (2014). From tagme to wat: A new entity annotator. In *Proceedings of the First International Workshop on Entity Recognition & Disambiguation*, ERD '14, pages 55–62, New York, NY, USA. ACM.

Smiley, D. (2013). Solr text tagger, text tagging with finite state transducers. `https://github.com/OpenSextant/SolrTextTagger`.

Tjong Kim Sang, E. F. and De Meulder, F. (2003). Introduction to the conll-2003 shared task: Language-independent named entity recognition. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003 - Volume 4*, CONLL '03, pages 142–147, Stroudsburg, PA, USA. Association for Computational Linguistics.

Wikipedia (2016a). List of Wikipedias - Wikipedia, the free encyclopedia. `https://en.wikipedia.org/wiki/List_of_Wikipedias`.

Wikipedia (2016b). Michael Jackson (disambiguation) - Wikipedia, the free encyclopedia. `https://en.wikipedia.org/wiki/Michael_Jackson_(disambiguation)`.

Appendices

Appendix A
Code examples: Manually written rules

```
private boolean looksLikeAName(Mention mention, Candidate c) {
    String text = mention.getText();
    boolean firstWordOfSentence = beginningOfSentence(mention.getStart());

    Pattern pattern = Pattern.compile("\\p{Lu}");
    Matcher  matcher = pattern.matcher(text);
    int nbrOfCapitalLetters = 0;
    while (matcher.find()) {
        nbrOfCapitalLetters++;
    }

    if(nbrOfCapitalLetters==0) {
        return false;
    }
    if (nbrOfCapitalLetters==1 && firstWordOfSentence
            && c.isOnlywords()) {
        return false;
    }
    return true;
}
```

Figure A.1: The function determining if a mention looks like a name, depending on casing and content.

```
public boolean matchIsInHeadline(Mention mention) {
    String line = getFullLine(mention);
    String[] parts = line.split(" ");
    int capitalizedWords = 0;
    int words = 0;
    for(String word: parts) {
        if(word.matches("\\p{Lu}.*")) {
            capitalizedWords++;
        }
        if(word.matches(".*\\p{L}.*")) {
            words++;
        }
    }
    float ratioInHeadline = 0.7f;
    return ((float)capitalizedWords)/words > ratioInHeadline;
}
```

Figure A.2: The function determining if a mention is part of a headline, depending on the capitalization of the line the mention is in.

```
public boolean keep(Candidate candidate, Mention mention) {
    String surfaceForm = candidate.getSurfaceForm();
    int suspicion = 0;
    int cutoff = 100;

    if (!looksLikeAName(mention, candidate)) {
        suspicion += 200;
    }
    if (candidate.isStopwords()) {
        suspicion += 200;
    }
    else if (candidate.isOnlywords() && isOneWord(candidate)) {
        suspicion += 100;
    }
    if (candidate.isGeneric()) {
        suspicion+=50;
    }
    if (matchIsInHeadline(mention) && candidate.isOnlywords()) {
        suspicion += 101;
    }
    if (!isDefaultSense(candidate)) {
        suspicion += 50;
    }
    if (candidate.getCount() < 15) {
        suspicion += 50;
    }
    if (candidate.getCount() < 5) {
        suspicion+=50;
    }
    if (candidate.isBold()) {
        suspicion-=50;
    }

    suspicion -= surfaceForm.length()/10;
    candidate.setSuspicion(suspicion);
    return suspicion < cutoff;
}
```

Figure A.3: The function that collects the rules.

INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | PRESENTATIONSDAG 2016-06-03

EXAMENSARBETE HERD - A Named Entity Recognizer and Disambiguator
STUDENT Anton Södergren
HANDLEDARE Pierre Nugues (LTH)
EXAMINATOR Elin Anna Topp (LTH)

Namnigenkänning och länkning i naturligt språk

POPULÄRVETENSKAPLIG SAMMANFATTNING **Anton Södergren**

För att samla stora mängder information om personer, platser och organisationer, behöver vi kunna analysera vanliga texter skrivna i naturligt språk. Detta arbete bidrar till det, genom att känna igen namn och länka dem till rätt Wikipedia-artikel.

För att kunna skapa system som kan svara på frågor, eller ge rätt svar vid sökningar, behöver vi analysera text: den största källan av information som finns tillgänglig. Eftersom det är en ostrukturerad och tvetydig informationskälla, behöver vi först transformera den till något som är lättare för datorer att förstå. Det här arbetet ämnar att göra text mer strukturerad, genom att känna igen namn i text, och länka varje namn till en artikel på Wikipedia, om en sådan artikel finns.

Det finns två stora svårigheter med namnigenkänning. Den första är att varje namn har flera olika möjliga lösningar. Det finns till exempel 21 personer vid namn “Michael Jackson” kända nog att ha sin egen Wikipedia-sida, och tusentals fler som inte har det. Den andra svårigheten är att avgöra om ett stycke text innehåller ett namn, vilket är lättare att förstå givet att det finns en artist, en amerikansk by, en telefonoperatör och en kinesisk bank som alla heter “Tomato”, vilket oftast inte är ett namn alls.

Mitt examensarbete har fokuserat på att skapa ett system för att känna igen namn och länka dem till en unik identifierare. Jag har skapat en algoritm som samlar idéer från olika state-of-the-art system, och utvärderat dem. Systemet skulle vara snabbt nog att kunna köras på hela Wikipedia inom en rimlig tidsram, och i kontrast med många andra system skulle det vara flerspråkigt. Dess effektivitet mättes på samma vis som ERD’14, en stor tävling där forskargrupper tävlade med sina system på samma dataset.

Systemet jag har byggt baseras på att analysera alla länkar mellan olika sidor i Wikipedia, och sedan räkna hur ofta en viss fras leder till en viss sida. På så vis kan vi samla alla olika namn och smeknamn en viss entitet har, och dessutom avgöra vilken länk som är vanligast för varje fras. Denna data laddas in i en databas och vid läsning av nya texter markeras alla fraser som finns i databasen.

Vilka fraser som ska behållas avgörs i flera steg. Det första är att med några enkla regler ta bort det minst troliga. Sedan körs en iterativ variant av algoritmen PageRank. Den försöker avgöra hur mycket två olika namn har gemensamt, genom att titta på om de har länkat till varandra i Wikipedia, eller nämnts i samma artikel tidigare. Systemet väljer sedan vem som menas med en fras, genom att ta den artikel med flest antal länkar för den frasen i Wikipedia.

Systemet nådde ett F1-resultat, vilket är ett genomsnitt av kvalitén för metoden, på 70% för engelska, vilket hade placerat det på en sjätteplats i ERD’14. Systemet finns även tillgängligt för svenska och franska.