BACHELOR THESIS
COMPUTING SCIENCE

RADBOUD UNIVERSITY

Effect of Surface Form Dictionary on Effectiveness in Entity Linking

Author:
Hermen van Westen
S4797620

First supervisor/assessor:
Dr. ir. F. Hasibi
F.Hasibi@cs.ru.nl

Second assessor:
Prof. dr. ir. A. P. de Vries
a.devries@cs.ru.nl

November 1, 2021

Abstract

This thesis aims to explore the effect of using ClueWeb annotations for candidate selection and disambiguation in entity linking. We used the Radboud Entity Linker (REL) to test various configurations, and discovered that using these annotations can have a significant impact on both recall and disambiguation. We think this research indicates that it is worthwhile to consider including ClueWeb annotations in the candidate selection part of entity linking systems.

Contents

Chapter 1

Introduction

Entity linking is a tool that helps people reading an article understand additional context or background without disrupting their reading flow. For example, someone reading an article on Wikipedia might hover over a specific event or person that is mentioned in the text, and get a better understanding of the article by doing so. We also use entity linking to improve both search results themselves and the way we can interact with these results. Finally, entity linking also plays a role in other research areas such as text summarization and question answering [1]. An example how entity linking is used can be found in Figure 1.1.

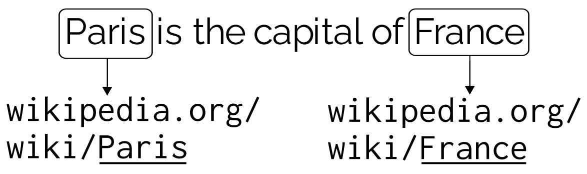

Paris is the capital of France
wikipedia.org/ wiki/Paris
wikipedia.org/ wiki/France

Figure 1.1: Example of entity linking

In this example we have a sentence that contains two mentions, which we then link to their appropriate Wikipedia page. We do this entity linking by first scanning the text and finding all the mentions, after which we can link them to their corresponding entries in the knowledge repository [1]. A knowledge repository is nothing more than a collection of entities that also contains other information such as relationships between entities and entity types. We use Wikipedia for this purpose in our research, since it has proved well-suited for this task [2, 3].

A common approach to implementing an entity linking system is to use a pipeline with three steps. This pipeline starts with *mention detection* to find mentions as discussed before. Then it does *candidate selection*, where it provides a list of candidate entities that each mention can be linked to. It finishes with *disambiguation*, where it decides on a single entity to link the mention to. We discuss these systems again in Section 2.2.

We use the Radboud Entity Linker (REL) [17] as the entity linker in our research, which also makes use of this pipeline. In particular, we focus on the candidate selection step of this pipeline. While REL normally uses a combination of Wikipedia, CrossWiki and YAGO to determine which entities are provided in the list of candidate entities at the end of the candidate selection step [16, 9], we are interested in the effect of replacing one or more of these with ClueWeb.

These ClueWeb annotations were automatically created by researchers at Google, and are based on data from ClueWeb09 and ClueWeb12, which contain web documents retrieved in 2009 and 2012. These annotations were named the Freebase Annotations of the ClueWeb Corpora, v1 (FACC1), and they exist for both ClueWeb09[1] and ClueWeb12[2].

1.1 Objectives

To be specific, the goal of this thesis is to answer the following questions:

- How does using ClueWeb annotations influence the candidate selection of an entity linker?
- How are the end-to-end entity linking results affected by incorporating these ClueWeb annotations?

1.2 Approach and Contributions

To find answers to these questions, we first have to process the ClueWeb data for use in REL, which means creating a mapping from the Freebase entities to Wikipedia entities. We then create various configurations that all use these annotations, which we then use in our entity linking system to see how candidate selection and end-to-end entity linking is affected.

We also develop an alternative implementation of candidate selection that uses less memory than the default implementation that REL provides. This was necessary since we ran into some memory issues when executing the program on our machine. This version does, however, lead to longer execution times. Our contributions then are as follows:

- We release a version of the candidate selection that has a lower memory footprint
- We show how using ClueWeb annotations affect the candidate entity selection of the entity linker

[1]`https://lemurproject.org/clueweb09/FACC1/`

[2]`https://lemurproject.org/clueweb12/FACC1/`

- We check how end-to-end entity linking results are affected by using ClueWeb annotations during candidate selection
- The code, alongside steps to recreate our experiments, can be found at `https://github.com/hvwesten/REL-experiments`.
- Our extended and commented version of REL can be found at `https://github.com/hvwesten/REL`.

1.3 Outline

The rest of this thesis is structured as follows. In Chapter 2, we provide some more information about entity linking and about our data. We also cover some related work in this chapter. Chapter 3 then covers the approach we took to find answers to our research questions, followed by the results of our work in Chapter 4. We then state our conclusions in Chapter 5, where we also touch on some future work. Finally, Appendix A contains some examples of the most important pieces of our code, Appendix B contains the steps to take to replicate our results, and Appendix C gives an example of output retrieved after evaluation.

Chapter 2

Background

In this chapter we will explain what entity linking is and how it is used in some machine learning tasks. We then discuss some relevant entity linking systems, after which we dive into the system that our research uses, focusing on the part that is most relevant to our research question. Next, we describe how we measure the performance of an entity linking system, after which we finish with an explanation of the data that we will be using in our research.

2.1 Entity Linking

2.1.1 Definition

In natural language processing (NLP) our goal is to make machines understand and use human-generated text. Being able to identify entities in text plays a big role in understanding what a text is about. Since mentions of entities can sometimes be ambiguous, we need some way to *link* these entities to their correct 'meaning'. Although humans can use their prior knowledge and the context to make this decision, machines tend to find this task relatively challenging. In entity linking (EL) we aim to solve this problem. The book 'Entity-oriented search' by K. Balog defines entity linking as

> "(...) the task of recognizing entity mentions in text and linking them to the corresponding entries in a knowledge repository".

This definition captures the idea behind entity linking very well, since our goal is to find a *link* for each mention in the text to its corresponding entity. The knowledge repository here is just a catalog of entities that a given mention can be linked to. A more formal definition of the entity linking task is: given a text d and a knowledge repository with a set of entities E, our goal is to find the mentions M_d in this text, and to map each mention $m \in M_d$ to its corresponding entity $e \in E$ in the knowledge base. This results in the set of annotations A_d for this specific text [1, 15].

2.1.2 Importance of Entity Linking

Once again, the reason that we want these annotated texts is to increase the understanding of machines from natural language text. We now discuss some specific natural language tasks that benefit from well-annotated text to show how the annotations provided by entity linking are used in practice.

One such task is *information extraction*, where text is processed to extract structured information such as entities and the relationships between them. These entities can often be ambiguous, which is why linking them to a knowledge base can be useful for disambiguation.

Another example of such a task is *information retrieval* in which we aim to retrieve documents that contain text related to a specific information need within a large collection of documents [12]. A straightforward example would be an internet search that returns several relevant web pages. Successful information retrieval systems then require annotations for both avoiding query ambiguity and determining the relevant web pages.

As a final example, we also use these annotations for automatic *knowledge base population*. New mentions that involve relations with entities already within the knowledge base then have to be linked using entity linking. These are just some examples among many, and we hope that this shows how entity linking is relevant for other natural language processing tasks [1, 15].

2.2 Entity Linking Systems

Several systems have been developed over the years that execute entity linking. Many of these systems use a three-step approach to generate the annotations. A text is first scanned to find the mentions (*mention detection*), after which several possible candidate entities are retrieved to which the mention could be linked (*candidate selection*). Finally, a decision is made in the third step as to which entity each mention finally gets linked to (*disambiguation*). An illustration of these steps can be found in Figure 2.1.

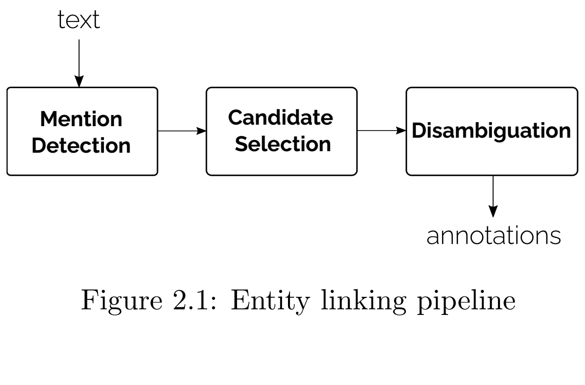


Figure 2.1: Entity linking pipeline

We now describe some entity linking systems that also follow a similar approach. The research discussed in this thesis can also be relevant for these systems since it affects the candidate selection step.

2.2.1 TAGME

One of the most popular entity linking systems is TAGME [5], with the original paper getting nearly 800 citations on Google Scholar at the time of writing. Although this system was originally designed to efficiently annotate short texts, it has also been used for longer text with great success.

TAGME uses a pipeline of three steps, namely parsing, disambiguation and pruning. The mentions and a set of candidate entities for each mention are retrieved in the parsing step, after which the best candidate is selected in the disambiguation step. The pruning step is then used to get rid of any annotations that are not meaningful [5, 8]. These steps are similar to the ones in the common approach we described earlier.

2.2.2 End-to-End Neural Entity Linking

The paper 'End-to-End Neural Entity Linking' [10] also describes an entity linking system using neural network-based approach. They describe their EL system as first generating all possible mentions and then giving each combination a score based on the context, encouraging the correct combination during training. Although this system does not describe a full seperate entity linking step, we still think our research can be relevant for this system.

2.2.3 CHOLAN

Another recently released entity linking system is CHOLAN [13], which also provides a modular approach to entity linking. Their paper uses an architecture consisting of mention detection, candidate generation and entity disambiguation, and employs the popular BERT transformer model [4] for both mention detection and disambiguation. This modular approach also corresponds to the three-step approach we described earlier.

2.3 The Radboud Entity Linker

In 2020 the Radboud Entity Linker was released. It aims to provide an entity linking system based on the latest developments in natural language processing research. By using a modular approach to creating such a system, they facilitate modifications and improvements. Since it is open-source, the code for it can be found on GitHub[1]. This sytem also uses a three-step approach, and is the one that we will mainly be using in our research.

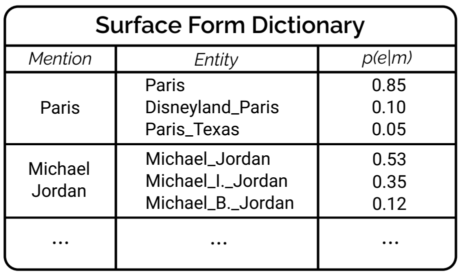
[1]`https://github.com/informagi/REL`

2.3.1 Candidate Selection in REL

The Radboud Entity Linker provides 7 candidate entities for each mention [6, 11, 17]. The first 3 are chosen based on their similarity to the context of the mention, but our research does not concern these. The other 4 are chosen based on a prior $P(e|m)$, which is a combination of $P_{\text{Wiki}}(e|m)$, calculated by counting how often each mention-entity combination occurs and dividing this by the total amount of combinations for this mention for both Wikipedia and Crosswiki [16], and $P_{\text{YAGO}}(e|m)$, calculated similarly except we only count each combination once (as in a uniform probablility) for YAGO [9]. These are then combined as follows:

$$P(e|m) = \min(1, P_{\text{Wiki}}(e|m) + P_{\text{YAGO}}(e|m) \tag{2.1}$$

This $P(e|m)$ is stored in a surface form dictionary together with the actual mentions and entities. A simplified example of such a surface form dictionary can be seen in Figure 2.2.



Surface Form Dictionary

Mention	*Entity*	*p(e|m)*
Paris	Paris	0.85
	Disneyland_Paris	0.10
	Paris_Texas	0.05
Michael Jordan	Michael_Jordan	0.53
	Michael_I._Jordan	0.35
	Michael_B._Jordan	0.12
...	...	...

Figure 2.2: Example of a surface form dictionary

A typical dictionary may contain up to a 100 entities per mention. Of these entities, the top 4 scoring entities get selected as the candidates for the mention. These candidates then get processed in the disambiguation step, which ultimately determines the actual entity that the mention will be linked to.

2.4 Evaluating an entity linking system

To assess an entity linking system, we compare the system-predicted annotations against a reference standard. We use several evaluation measures, namely *precision*, *recall* and the *F-measure*. Since we are interested in the performance over several documents, we calculate these measures either by aggregating across mentions (micro-averaged) or documents (macro-averaged). We use A_D to denote the set of generated annotations for a set of D documents, and $\hat{A}_D$ for the reference annotations for these D documents.

The micro-averaged precision and recall are then calculated as follows:

$$P_{\text{micro}} = \frac{|A_D \cap \hat{A}_D|}{|A_D|}, \qquad R_{\text{micro}} = \frac{|A_D \cap \hat{A}_D|}{|\hat{A}_D|},$$

where $|A_D \cap \hat{A}_D|$ refers to the number of correctly linked entity mentions. The macro-averaged versions of these scores are as follows:

$$P_{\text{macro}} = \frac{1}{|D|}\sum_{d \in D} \frac{|A_d \cap \hat{A}_d|}{|A_d|}, \qquad R_{\text{macro}} = \frac{1}{|D|}\sum_{d \in D} \frac{|A_d \cap \hat{A}_d|}{|\hat{A}_d|}.$$

The F-measure (using either the micro or macro precision and recall scores) is then calculated like so:

$$F_1 = \frac{2PR}{P+R}.$$

Although these scores show the performance of our entity linking system on a specific set of documents, we need some way to compare the performance of several systems against each other. For this purpose we use an evaluation platform called GERBIL [14] that uses standardized experiments to provide this benchmarking solution. It can be used either locally or by using the web version[2] and connecting this to your own entity linking system via an API. We are interested in the prediction being a precise match of the true entity, which is known as strong matching. Furthermore, we only consider mentions that actually have entities in our knowledge repository (InKB).

2.5 The ClueWeb annotations

The data we use in our research consists of annotations made by research at Google for both ClueWeb09[3] and ClueWeb12[4]. The ClueWeb09 data is split up into 10 parts, and each of these parts contains several folders.

These folders then contain `tsv` files, where each line contains a mention, some information about its location and posterior (the probablity that this mention will occur given the mention and the context), and the entity itself. The entity is denoted by a Freebase identifier, which is from a deprecated corpus that we no longer use (more on this in Section 3.1). An example line from ClueWeb09 is shown below:

```
Paris 21089 21092 0.99 0.0006 /m/0k3xf
```

[2]`https://gerbil.aksw.org/gerbil/config`
[3]`https://lemurproject.org/clueweb09/FACC1/`
[4]`https://lemurproject.org/clueweb12/FACC1/`

The data for ClueWeb12 looks exactly the same, although it is split up into 20 parts instead of 10. We use this ClueWeb data mainly since it seems to cover similar entities as CrossWiki, but we are unsure how this works when actually using it for candidate selection and entity linking.

Chapter 3

Approach

In this chapter we will describe the approach and methods that we used in our research. We start by describing how we processed the ClueWeb data in such a way that it can be used in the Radboud Entity Linker. Next, we illustrate how we actually use this data to calculate the prior that we need. We then describe how we use this prior in our entity linking system, and finish with a description of our implementation of candidate selection that makes use of a database to store the data.

3.1 Processing the ClueWeb annotations

Before we could actually use our ClueWeb annotations, we first had to convert the Freebase entities that it contains to Wikipedia entities, because that is what REL uses. To do so, we used the latest mapping from Freebase to Wikidata from Google[1], and mapped these Wikidata entities to Wikipedia titles using a tool called *wikimapper*[2]. This tool can be used with various Wikipedia indices, but we used it with the index for Wikipedia 2019 that is provided in this repository. Since the mapping from Wikidata IDs to Wikipedia titles is not unique due to redirects, we get multiple possible Wikipedia titles for each ID. Our approach was to use the most common title, but other approaches can be used.

Our implementation (named `FreebaseWikipediaMapper`) can be found in the `mapping` folder in the repository for the experiments[3]. It works by going through all the combinations from the Freebase to Wikidata file, looking these Wikidata entities up using the `wikimapper`, and finally creating a dictionary full of Freebase mentions and their Wikipedia entities. We make sure that all entries in this dictionary are actual Wikipedia entities by using an auxiliary function provided in REL.

[1] `https://developers.google.com/freebase/`

[2] `https://github.com/jcklie/wikimapper`

[3] `https://github.com/hvwesten/REL-experiments/tree/main/mapping`

To use this data in our calculation of the prior $p(e|m)$ in REL, we then have to count how often each combination of mention and entity occurs in the data. We save this data in a `json` file so that we can more easily use this data across multiple experiments. A short excerpt from this file looks like this:

```
{
    "Paris Saint-Germain": {
        "Paris_Saint-Germain_F.C.": 2260,
        "Paris_Saint-Germain_Academy": 17,
        "Paris_Saint-Germain_Féminines": 23,
        "Paris_Saint-Germain_Rugby_League": 1,
        "Colborne_Parish,_New_Brunswick": 121,
        "Paris_Saint-Germain_F.C.": 1,
        "Paris_FC": 1,
        "St._George's_Parish,_Bermuda": 2,
    },
    "Paris France": {
        "Paris": 177
        "Paris_Saint-Germain_F.C.": 2,
        "Belleville,_Paris": 17,
        "Kingston_Parish": 21,
    },
    ...
}
```

We executed this conversion on both the ClueWeb09 and ClueWeb12 annotations, and saved them in a special `save_folder` so that they can be easily loaded in future experiments. This means that we don't have to repeat the same conversion every experiment, which saves a lot of time since this conversion can take up to 2 hours to complete. This is also the reason why we decided not to implement this conversion within REL itself, but instead place it alongside the rest of the code in our experiments.

As a sidenote, we first tried to implement this conversion using a database, but we soon discovered that this would take too long. The candidate selection part of the program has to retrieve data thousands of times, which can take a long time when using a database. This is why we decided to keep the mappings in memory instead.

3.2 Computing the prior

To see how using ClueWeb09 and ClueWeb12 annotations affects candidate selection we decided to do our experiments using just ClueWeb09 and the combination of both, which we call ClueWeb09+12 . We use these counts to compute a new P_{Wiki}, as described in Equation 2.1. Specifically, P_{Wiki} is computed by summing up counts from Wikipedia, CrossWiki and ClueWeb annotations (when necessary, depending on the setting). We experiment with seven different configurations. These provide various possibilities in which one could use the annotations. For brevity, we represent the counts from Wikipedia with *W*, from CrossWiki with *CW*, from YAGO with *Y*, from ClueWeb09 with *C9*, and from ClueWeb-09+12 with *C9+C12*:

1. *C9 (+ C12) + W + Y*

   This configuration replaces CrossWiki with the ClueWeb annotations, either ClueWeb09 or ClueWeb09+12.

2. *CW + C9 (+ C12) + Y*

   This configuration replaces Wikipedia with the ClueWeb annotations, either from ClueWeb09 or ClueWeb09+12.

3. *CW + W + C9 (+ C12)*

   This configuration replaces YAGO with the ClueWeb annotations, either ClueWeb09 or ClueWeb09+12.

4. *C9 (+ C12)*

   This configuration uses only ClueWeb09 or ClueWeb09+12.

5. *Y + C9 (+ C12)*

   This configuration only uses ClueWeb09 or ClueWeb09+12, together with YAGO.

6. *CW + C9 (+ C12)*

   This configuration only uses CrossWiki and ClueWeb09 or ClueWeb09+12.

7. *CW + W + Y + C9 (+ C12)*

   This configuration combines all our datasets by extending the baseline with either ClueWeb09 or ClueWeb09+12.

In total, this gives us 14 different results, which we will explore in Section 4.2. Not only are we interested in how each of these configurations performs, but also what the difference is between using just ClueWeb09 versus combining both in ClueWeb09+12. The code that we used for this calculation can be found in code snippet 1 and 3 from Appendix A.

The main thought behind our implementation was to facilitate replacement of our datasets with other data, although some of our configurations required a hard-coded solution.

3.3 Using ClueWeb for candidate selection

Once we have the prior that we want to use, we continue, and use this for candidate selection. The steps are relatively straightforward, and are described in Appendix B. To complete the candidate selection step, we need Wikipedia embeddings to create the 3 other candidate entities based on similarity. This step is not required, since the authors of REL already created them and they are provided in their repository. We did not execute this step either since executing it would take a lot of compute power, but we include it anyways for the sake of completeness.

The next step is to generate the training and testing files. This is where we add the top 4 ranked entities from the surface dictionary. With this done, we can continue to the disambiguation step.

The code for training and evaluating our EL system can be found in code snippet 5 from Appendix A. Two important things to note here are the location of our data (which is described in Appendix B), and the location of our resulting models, which in our case was

```
base_url/wiki_2019/baseline_cw_<No.experiment>.
```

We ended up with 15 different models (including the baseline model), although not all of them were fully trained. We either trained the models for 5 epochs in order to look at the recall scores, or trained them fully when we wanted to use them for the disambiguation step.

The recall scores are retrieved by evaluating a model (either fully or non-fully trained), since doing so prints these scores. The F1-scores are also retrieved in this way, although these are only relevant when the model has been fully trained. An example of the output is shown in Appendix C.

3.4 Using a database for candidate selection

The first time that we started processing the ClueWeb data, we ran into memory problems. The default implementation of the prior calculation seemed to run out of memory on our machine, requiring more than 16GB of memory to finish the calculations. Since we still wanted to continue our experiments, we decided to implement an alternative version that uses a database to store all the mentions and their entities.

We used SQLite[4] to store the data. The Wikipedia and CrossWiki counts are retrieved using the same method that REL uses, but this time they are also saved in a database. The ClueWeb counts are read from the file described in Section 3.1, and are also saved in a database.

They are saved in `base_url/counts/` as `wiki.db` and `custom.db` , respectively, although these locations can easily be changed. Both these databases are filled in batches of 500 000 to keep the memory usage as low as possible, and an index is used to make future retrievals as quick as possible. These counts are then read in batches, and are used to calculate the actual $p(e|m)$ values. These values are kept in memory until they get saved in a different database (which the default REL already does).

The most important thought behind the implementation is the batch insertion and retrieval, which you can find in code snippet 2 and 4 from Appendix A. It is important to note that, while our implementation is fully functional, we did not spend a lot of time optimizing it, since soon after we swapped to a new machine for efficiency reasons.

[4]`https://sqlite.org/index.html`

Chapter 4

Results

In this chapter we show the results of experiments. We start by looking at the memory usage of our candidate selection implementation. Unfortunately we ended up not using this implementation very much. We then look at the recall scores that we got when using the ClueWeb09 and ClueWeb12 datasets. Finally, we check whether these recall scores actually resulted in more effective entity disambiguation.

4.1 Memory Usage

We compared the default $p(e|m)$ calculation (denoted by *no_db*) against two versions of our implementation described in Section 3.4, one of which re-creates the database every run (*db*), while the other preloads the data (*db + preload*). This was tested while calculating the configuration *W + CW + C9 (+ C12)*, although it could be possible that a bigger improvement can be found using a different configuration. We also compare the difference between using ClueWeb09 and ClueWeb09+12.

Dataset	Version	Peak Memory Usage	Elapsed Time
ClueWeb09+12	*no_db*	17.6GB	6:40
	db	17.5GB	12:18
	db + preload	17.4GB	11:32
ClueWeb09	*no_db*	16.9GB	6:32
	db	16.0GB	12:17
	db + preload	16.0GB	10:29

Table 4.1: Memory usage and execution time

As can be seen in Table 4.1, our implementation of the $p(e|m)$ calculation indeed uses less memory, but trades this off for a longer execution time. Since ClueWeb09+12 contains many more mentions than ClueWeb09, it seems to need more memory, which is to be expected.

They do, however, both take approximately the same time to finish, which indicates that the difference lies mainly in storage, not processing. These numbers are achieved on a system with a Ryzen 5 5600X processor and 32GB of memory. Since the default version is much quicker on this machine, we decided to use this implementation for the rest of our experiments. Note that this version did not work on our previous machine, which is why we decided to create a database version in the first place, as described in Section 3.4. We expect that memory usage can be reduced even more by fine-tuning the insertion and retrieval of data from the database, but, as we mentioned before, we did not spent as much time optimizing our implementation.

4.2 Recall

As described in Section 3.2, we experimented with seven different configurations. The recall scores of these are compared on seven datasets, namely AIDA-A and AIDA-B, which are hand-annotated datasets based on the CoNLL 2003 data [9], AQUAINT, which contain news articles with linkable mentions from a news corpus that, among others, includes the New York Times, MSNBC, with linkable mentions from news articles from the MSNBC, ACE2004, which is an annotated subset of the documents used in the ACE20004 Coreference documents [7], and the ClueWeb and Wikipedia datasets. The scores are shown in Table 4.2.

Recall	AIDA-A	AIDA-B	ACE2004	AQUAINT	CLUEWEB	MSNBC	WIKIPEDIA
CW+W+Y (baseline)	*94.8*	**96.3**	*88.3*	*93.4*	88.6	97.2	**86.2**
C9+W+Y	94.2	*94.9*	86.4	90.1	89.9	96.0	82.3
CW+C9+Y	*94.8*	**96.3**	*88.3*	93.3	*91.6*	*97.7*	84.5
CW+W+C9	92.8	94.6	*88.3*	93.3	89.7	97.1	83.8
C9	84.4	86.2	82.1	75.9	81.7	82.7	60.5
Y+C9	92.5	93.3	84.8	71.7	87.4	93.9	72.4
CW+C9	92.7	94.4	87.9	92.6	89.1	96.2	82.8
CW+W+Y+C9	**95.1**	**96.3**	**88.7**	**93.5**	**92.0**	**97.9**	*85.3*

Table 4.2: Recall ClueWeb09

Our best results were achieved by combining Wikipedia, CrossWiki, ClueWeb09 and YAGO. This configuration achieved higher scores than our baseline on all datasets except on Wikipedia itself. As expected, including CrossWiki helps with the recall on the Wikipedia dataset. Interestingly, using ClueWeb instead of Wikipedia already results in relatively good recall scores. When also including the ClueWeb12 counts, we see similar improvements as shown in Table 4.3.

Recall	AIDA-A	AIDA-B	ACE2004	AQUAINT	CLUEWEB	MSNBC	WIKIPEDIA
CW+W+Y (baseline)	94.8	**96.3**	*88.3*	**93.4**	88.6	**97.2**	**86.2**
C9+C12+W+Y	94.4	94.9	86.4	90.4	89.8	95.6	82.2
CW+C9+C12+Y	*94.9*	95.8	*88.3*	92.8	*91.2*	96.9	84.5
CW+W+C9+C12	93.4	94.8	*88.3*	**93.4**	89.5	*97.1*	83.9
C9+C12	85.1	86.6	82.5	76.2	82.2	82.8	61.1
Y+C9+C12	92.7	93.4	84.8	72.6	87.5	93.6	72.6
CW+C9+C12	93.1	94.6	87.9	92.7	89.1	96.2	83.0
CW+W+Y+C9+C12	**95.3**	*95.9*	**88.7**	*93.1*	**91.3**	*97.1*	*85.1*

Table 4.3: Recall ClueWeb09+12

As you can see, the configuration that uses all the datasets still performs pretty well, getting either the highest or second highest recall scores. Still, it does not perform our baseline model as much as the configuration using ClueWeb09 did, most likely because of domination of highly frequent entities.

4.3 Entity Disambiguation

We only trained the models with the best recall scores, which included *CW+W+C9*, *CW+C9+Y*, *CW+W+Y+C9* and *CW+W+Y+C9+C12*. We evaluated their performance using the GERBIL platform [14] that we covered in Section 2.4. To be specific, we set up a local entity linking system and used this in the online platform[1]. The results can be found in Table 4.4.

Macro F1 / **Micro F1**	AIDA-B	Derczynski	KORE50	MSNBC	N3-REUTERS-128	N3-RSS-500	OKE-2015	OKE-2016	**Average**
Baseline	**82.9**	62.3	*54.4*	*86.3*	58.2	*61.7*	*64.0*	**67.0**	*67.1*
	84.0	62.0	54.0	*85.8*	**64.9**	64.1	64.3	**67.3**	*68.3*
CW+W+C9	77.9	63.2	47.6	85.0	58.3	59.5	**65.0**	*66.6*	65.4
	77.4	*62.5*	50.5	84.0	62.9	61.3	**65.3**	66.9	66.4
CW+C9+Y	82.1	*63.4*	48.6	85.8	58.2	*61.7*	*64.0*	61.5	65.7
	82.5	61.1	50.5	85.4	63.7	*65.0*	*64.4*	63.2	67.0
CW+W+Y+C9	*82.6*	62.9	53.2	**87.1**	**58.5**	*61.7*	63.8	65.0	66.9
	83.1	61.5	*55.4*	**86.0**	*64.1*	*65.0*	63.5	66.1	68.1
CW+W+Y+C9+C12	80.9	**65.3**	**60.2**	84.8	*58.4*	**61.9**	62.7	65.9	**67.5**
	80.5	**63.5**	**62.5**	84.8	63.4	**65.7**	63.2	*67.1*	**68.8**

Table 4.4: ED results on GERBIL

[1] https://gerbil.aksw.org/gerbil/config

It seems like we get the most effective disambiguation when we include all of the datasets from the baseline (Wikipedia, CrossWiki and YAGO). Including ClueWeb09 and ClueWeb12 gives us our best result, getting the best average Macro- and Micro-F1 scores among all configurations. The biggest improvement seems to occur on the Derczynski and KORE50 datasets. More details about the datasets used for entity disambiguation can be found in [7].

4.4 Analysis

When considering our initial goal of seeing how using ClueWeb annotations influences candidate selection and end-to-end entity linking, we see that including ClueWeb09 annotations results in good recall and therefore candidate selection. This, however, did not directly translate into good ED results, most likely because this dataset does not cover many of the entities included in the standard experiments on GERBIL.

When including ClueWeb09+12, we do not see such large improvements in the recall scores, but we do get our best ED results, and thereby our best end-to-end entity linking system. We can safely say that including some combination of ClueWeb data (most likely ClueWeb12 or ClueWeb09+12) in the candidate selection step actually results in better candidate selection and entity linking.

Chapter 5

Conclusions

In this thesis we created a mapping from Freebase entities to Wikipedia titles. We tried saving this mapping in a database, but this was not efficient enough for our purposes. The mapping was used to process the ClueWeb09 and ClueWeb12 annotations, which resulted in our ClueWeb09 and ClueWeb09+12 counts. These counts were used in the computation of the prior in our entity linking system, in 14 different configurations. We also developed an implementation of candidate selection that uses a database to store the data, but we ended up not using this very much in our research.

Most of our the configurations showed high recall scores. This means that candidate selection definitely benefits when including the ClueWeb annotations, which answers the first question we set out to ask at the beginning of this thesis. With regard to entity disambiguation, our efforts were not as successful. We only have one configuration that outperforms the baseline model, and this configuration only extends the baseline with ClueWeb09+12. We did succeed in showing how various configurations influence the ED results and thereby our entity linking system, which answers the second question that we posed at the start of this thesis.

5.1 Future Directions

Creating an entity linking system using a different dataset involves many steps and decisions. This means that there are several directions in which our research can be extended.

To start, we think that our database implementation can be improved upon. Although we managed to reduce the memory usage by around 1GB, we expect that it can be reduced even more, either by using something other than SQLite, or by improving upon the insertion and retrieval of data from the database. We also hope that the execution time can be reduced, since our implementation sometimes take twice as long as the default version, but we are not sure if this is possible.

We also think that more research can be done about the exact reason for the difference in performance between ClueWeb09 and ClueWeb09+12. Although we guessed at the reason for this effect, we expect that finding the exact reason can help in maximizing the performance of an entity linking system that uses ClueWeb annotations.

Finally, we think our experiments should be repeated with using just the ClueWeb12 annotations. This will help answer the issue mentioned above, but will probably also result in an entity linking system that is perhaps just as effective as the configuration that uses both ClueWeb09 and ClueWeb12 while using less data.

Bibliography

[1] Krisztian Balog. *Entity-Oriented Search*, volume 39 of *The Information Retrieval Series*. Springer, 2018.

[2] Razvan Bunescu and Marius Paşca. Using Encyclopedic Knowledge for Named Entity Disambiguation. In *11th Conference of the European Chapter of the Association for Computational Linguistics*, Trento, Italy, April 2006. Association for Computational Linguistics.

[3] Silviu Cucerzan. Large-Scale Named Entity Disambiguation Based on Wikipedia Data. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 708–716, Prague, Czech Republic, June 2007. Association for Computational Linguistics.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *CoRR*, abs/1810.04805, 2018.

[5] Paolo Ferragina and Ugo Scaiella. TAGME. In *Proceedings of the 19th ACM international conference on Information and knowledge management - CIKM '10*. ACM Press, 2010.

[6] Octavian-Eugen Ganea and Thomas Hofmann. Deep Joint Entity Disambiguation with Local Neural Attention. *CoRR*, abs/1704.04920, 2017.

[7] Zhaochen Guo and Denilson Barbosa. Robust Named Entity Disambiguation with Random Walks. *Semantic Web*, 9(4):459–479, June 2018.

[8] Faegheh Hasibi, Krisztian Balog, and Svein Erik Bratsberg. On the reproducibility of the TAGME Entity Linking system. In *roceedings of 38th European Conference on Information Retrieval*, ECIR '16, pages 436–449. Springer, 2016.

[9] 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 *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 782–792, Edinburgh, Scotland, UK., July 2011. Association for Computational Linguistics.

[10] Nikolaos Kolitsas, Octavian-Eugen Ganea, and Thomas Hofmann. End-to-End Neural Entity Linking. *CoRR*, abs/1808.07699, 2018.

[11] Phong Le and Ivan Titov. Improving Entity Linking by Modeling Latent Relations between Mentions. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1595–1604, Melbourne, Australia, July 2018. Association for Computational Linguistics.

[12] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, Cambridge, UK, 2008.

[13] Manoj Prabhakar Kannan Ravi, Kuldeep Singh, Isaiah Onando Mulang, Saeedeh Shekarpour, Johannes Hoffart, and Jens Lehmann. CHOLAN: A Modular Approach for Neural Entity Linking on Wikipedia and Wikidata. *CoRR*, abs/2101.09969, 2021.

[14] Michael Röder, Ricardo Usbeck, and Axel-Cyrille Ngonga Ngomo. GERBIL - Benchmarking Named Entity Recognition and Linking consistently. *Semantic Web*, 9(5):605–625, 2018.

[15] Wei Shen, Jianyong Wang, and Jiawei Han. Entity Linking with a Knowledge Base: Issues, Techniques, and Solutions. *IEEE Transactions on Knowledge and Data Engineering*, 27(2):443–460, February 2015.

[16] Valentin I. Spitkovsky and Angel X. Chang. A Cross-Lingual Dictionary for English Wikipedia Concepts. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*, pages 3168–3175, Istanbul, Turkey, May 2012. European Language Resources Association (ELRA).

[17] Johannes M. van Hulst, Faegheh Hasibi, Koen Dercksen, Krisztian Balog, and Arjen P. de Vries. REL: An Entity Linker Standing on the Shoulders of Giants. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '20. ACM, 2020.

Appendix A

Code Examples

We show some of the more important code snippets here that show how we load the counts and how we use these counts to calculate $p(e|m)$.

```
def __load_counts(self, custom_add):
"""
Updates mention/entity for Wiki with these additional counts
:return:
"""
for mention, entity_dict in custom_add.items():
    if mention not in self.wiki_freq:
        self.wiki_freq[mention] = {}

    for entity, count in entity_dict.items():
        # Preprocess the entity name
        ent_name = self.wikipedia.preprocess_ent_name(entity)

        if ent_name in (self.wikipedia.
        wiki_id_name_map["ent_name_to_id"]):
            if mention not in self.mention_freq:
                self.mention_freq[mention] = 0
            self.mention_freq[mention] += count

            ent_name = ent_name.replace(" ", "_")

            if ent_name not in self.wiki_freq[mention]:
                self.wiki_freq[mention][ent_name] = 0
            self.wiki_freq[mention][ent_name] += count
```

Code Snippet 1: Loading counts from a given file (without a database)

```
def __load_custom_db(self, custom=None):
    custom_list = []
    db = sqlite3.connect(self.custom_db_url)
    c = db.cursor()
    n = 0

    for mention, entity_dict in custom.items():
        n += 1
        if n % 500000 == 0:
            c.execute("BEGIN TRANSACTION;")
            c.executemany('''INSERT INTO
            custom_counts(mention, entity, count) VALUES (?, ?, ?)
            ON CONFLICT(mention, entity) DO UPDATE SET count=count + ?
            ''', custom_list)
            c.execute("COMMIT;")
            custom_list = []

        mentions = []
        entities = []
        counts = []

        for entity, count in entity_dict.items():
            # Process wikipedia title
            ent_name = self.wikipedia.preprocess_ent_name(entity)

            # Only add the mention if it is in the KB
            if ent_name in (self.wikipedia.
                            wiki_id_name_map["ent_name_to_id"]):
                ent_name = ent_name.replace(" ", "_")
                mentions.append(mention)
                entities.append(ent_name)
                counts.append(count)

        temp = [(m, e, c1, c2) for m, e, c1, c2 in zip(mentions,
                entities, counts, counts)]
        custom_list.extend(temp)

    # Insert last few mentions into database
    c.execute("BEGIN TRANSACTION;")
    c.executemany('''INSERT INTO custom_counts(mention, entity, count)
    VALUES (?, ?, ?) ON CONFLICT(mention, entity) DO UPDATE SET
    count=count + ? ''', custom_list)
    c.execute("COMMIT;")
    c.close()
    db.close()
```

Code Snippet 2: Loading counts from a given file (with a database)

```python
def compute_wiki(self, special_case=False, custom_main=None,
                custom_add=None):
    if special_case:
        print(f"You selected the special option: "{special_case}")
    if special_case == "only_clueweb":
        self.__load_counts(custom_main)
    elif special_case == "only_crosswiki":
        self.__cross_wiki_counts()
    elif special_case == "all":
        self.__wiki_counts()
        self.__cross_wiki_counts()
        self.__load_counts(custom_main)
    else:
        if custom_main:
            self.__load_counts(custom_main)
        else:
            self.__wiki_counts()
        if custom_add:
            self.__load_counts(custom_add)
        else:
            self.__cross_wiki_counts()

    # Step 1: Calculate p(e|m) for wiki.
    print("Filtering candidates and calculating p(e|m) values"
          "for Wikipedia.")
    for ent_mention in self.wiki_freq:
        if len(ent_mention) < 1:
            continue

        ent_wiki_names = sorted(
        self.wiki_freq[ent_mention].items(), key=lambda kv: kv[1],
            reverse=True
        )
        # Get the sum of at most 100 candidates, but less if less
        # are available.
        total_count = np.sum([v for k, v in ent_wiki_names][:100])

        if total_count < 1:
            continue
        self.p_e_m[ent_mention] = {}
        for ent_name, count in ent_wiki_names:
            self.p_e_m[ent_mention][ent_name] = count / total_count
        if len(self.p_e_m[ent_mention]) >= 100:
            break
        del self.wiki_freq
```

Code Snippet 3: Creating the final $p(e|m)$ (without a database)

```python
def compute_wiki_with_db(self):
    self.wiki_db_url =  os.path.join(self.base_url, "counts/wiki.db")

    self.__create_wiki_db()
    self.__wiki_counts(using_database=True)
    self.__cross_wiki_counts(using_database=True)
    self.__create_wiki_index()

    db = sqlite3.connect(self.wiki_db_url)
    c = db.cursor()
    d = db.cursor()

    c.execute('''SELECT DISTINCT mention FROM wiki_counts''')

    num_mentions = 0
    batch_size = 50000

    while True:
        batch = c.fetchmany(batch_size)

        if not batch:
            break

        for row in batch:
            num_mentions += 1

            ent_mention = row[0]
            if len(ent_mention) < 1:
                continue

            d.execute('''SELECT entity, count FROM wiki_counts
            WHERE mention=? ORDER BY count DESC''', (ent_mention, ))
            ent_wiki_names = d.fetchall()

            # Get the sum of at most 100 candidates, but less if
            # less are available.
            total_count = np.sum([v for k, v in ent_wiki_names][:100])

            if total_count < 1:
                continue

            self.p_e_m[ent_mention] = {}

            for ent_name, count in ent_wiki_names:
                self.p_e_m[ent_mention][ent_name] = count / total_count

                if len(self.p_e_m[ent_mention]) >= 100:
                    break
```

Code Snippet 4: Creating the final $p(e|m)$ (with a database)

```
from REL.training_datasets import TrainingEvaluationDatasets
from REL.entity_disambiguation import EntityDisambiguation

base_url = "/home/hvwesten/Projects/thesis/data/"
wiki_version = "wiki_2019"

datasets = TrainingEvaluationDatasets(base_url, wiki_version).load()

model_pth = f"{base_url}/{wiki_version}/baseline_cw_5/model
config = {
    "mode": "train",
    "model_path": model_pth,
}

model = EntityDisambiguation(base_url, wiki_version, config)

if config["mode"] == "train":
    model.train(
       datasets["aida_train"],
       {k: v for k, v in datasets.items() if k != "aida_train"}
)
else:
    model.evaluate({k: v for k, v in datasets.items()
            if "train" not in k})
```

Code Snippet 5: Code for training and evaluating our EL system

Appendix B

Recreating the experiments

This project uses our modified REL to conduct the experiments [1]. We mostly follow the tutorials[2] given in the original REL while using our own modified version of REL.

To recreate the experiments, you should first do the setup described in these tutorials. You can then download the ClueWeb09 and ClueWeb12 annotations, and place them in the `base_url` folder alongside `wiki_2019/` and generic/. Your folder structure should look something like this:

```
base_url
|----generic/
|----wiki_2019/
|----ClueWeb09/
|---- ClueWeb09_English_1.tgz
|---- ...
|----ClueWeb12/
|---- ClueWeb12_English_1.tgz
|---- ...
```

You can then execute the following steps:

0) Generate ClueWeb counts with `00_clueweb_to_json.py`.
1) Create $p(e|m)$ in one of two ways:

   a) Using memory, with `01a_generate_pem_clueweb.py`

   b) Using the database, with `01b_generate_pem_clueweb_using_DB.py`

2) (Not Required) Re-create embeddings with `02_train_embeddings.py`
3) Generate the training and test files with `03_generate_train_test_files.py`
4) Train or evaluate with `04_train_eval.py`
5) Execute the full end-to-end entity linking pipeline with `05_e2e_entity_linking.py`

[1]https://github.com/hvwesten/REL/tree/master/REL

[2]https://github.com/informagi/REL/tree/master/tutorials

We give an overview of the steps in figure B.1. The main point of note is that the result of step 1 and 2 both get saved in the same file, which then gets used in the other steps.

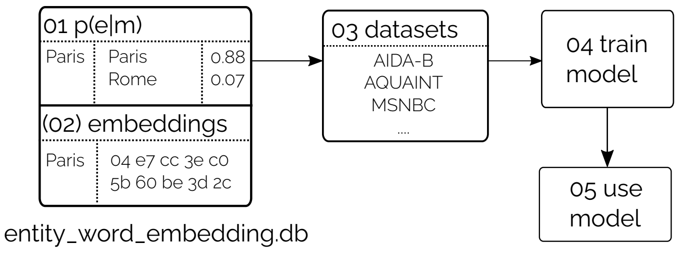


Figure B.1: A diagram of the implementation steps

These steps can also be found in our repository for the experiments [3]. The code itself also contains comments that help in understanding and executing the steps mentioned above.

[3]https://github.com/hvwesten/REL-experiments

Appendix C

Output of evaluation in REL

This is an example of the printout after executing the evaluation step.

```
Loading aida_train
Loading aida_testA
Loading aida_testB
Loading wned-ace2004
Loading wned-aquaint
Loading wned-clueweb

Loading wned-msnbc
Loading wned-wikipedia
model_pth:
/home/hvwesten/Projects/thesis/data//wiki_2019/baseline_cw_9/model
No LR model found, confidence scores ED will be set to zero.
Loading model from given path:
/home/hvwesten/Projects/thesis/data//wiki_2019/baseline_cw_9/model
Recall for aida_testA: 0.9509496973491964
----------------------------------------------
Recall for aida_testB: 0.9632107023411371
----------------------------------------------
Recall for wned-ace2004: 0.8871595330739299
----------------------------------------------
Recall for wned-aquaint: 0.9353507565337001
----------------------------------------------
Recall for wned-clueweb: 0.9199630314232902
----------------------------------------------
Recall for wned-msnbc: 0.9785276073619632
----------------------------------------------
Recall for wned-wikipedia: 0.8527670908552827
----------------------------------------------
```

```
Micro F1: 0.8816530995616781, Recall: 0.8816530995616781,
Precision: 0.8816530995616781
aida_testA None
Total NIL: 0
----------------------------------
Micro F1: 0.8900780379041249, Recall: 0.8900780379041249,
Precision: 0.8900780379041249
aida_testB None
Total NIL: 0
----------------------------------
Micro F1: 0.8732394366197183, Recall: 0.8443579766536965,
Precision: 0.9041666666666667
wned-ace2004 None
Total NIL: 17
----------------------------------
Micro F1: 0.8679245283018867, Recall: 0.8541953232462174,
Precision: 0.8821022727272727
wned-aquaint None
Total NIL: 23
----------------------------------
Micro F1: 0.7734024339456759, Recall: 0.772365988909427,
Precision: 0.7744416643499212
wned-clueweb None
Total NIL: 29
----------------------------------
Micro F1: 0.912442396313364, Recall: 0.911042944785276,
Precision: 0.9138461538461539
wned-msnbc None
Total NIL: 2
----------------------------------
Micro F1: 0.709797776285352, Recall: 0.7037585084344481,
Precision: 0.7159415926539214
wned-wikipedia None
Total NIL: 115
----------------------------------
```