Entity Linking

Kjetil Møkkelgjerd

Master of Science in Computer Science
Submission date: June 2017
Supervisor: Herindrasana Ramampiaro, IDI

Norwegian University of Science and Technology
Department of Computer Science

Abstract

Entity linking may be of help to quickly supplement a reader with further information about entities within a text by providing links to a knowledge base containing more information about each entity, and may thus potentially enhance the reading experience. In this thesis, we look at existing solutions, and implement our own deterministic entity linking system based on our research, using our own approach to the problem.

Our system extracts all entities within the input text, and then disambiguates each entity considering the context. The extraction step is handled by an external framework, while the disambiguation step focuses on entity-similarities, where similarity is defined by how similar the entities' categories are, which we measure by using data from a structured knowledge base called DBpedia. We base this approach on the assumption that similar entities usually occur close to each other in written text, thus we select entities that appears to be similar to other nearby entities.

Experiments show that our implementation is not as effective as some of the existing systems we use for reference, and that our approach has some weaknesses which should be addressed. For example, DBpedia is not an as consistent knowledge base as we would like, and the entity extraction framework often fail to reproduce the same entity set as the dataset we use for evaluation. However, our solution show promising results in many cases.

Sammendrag

Entitetlenking kan være til hjelp for å gi en leser enkel tilgang til supplerende informasjon om entiteter in en tekst ved å oppgi lenker til en kunnskapsbase som har mer informasjon om hver enkelt entitet, noe som potensielt kan forbedre leseopplevelsen. I denne avhandlingen ser vi på eksisterende løsninger, og implementerer vårt eget deterministiske entitetlenkingssystem på bakgrunn av vår forskning, ved å bruke vår egen tilnærming til problemet.

Systemet vårt henter ut alle entiteter i en gitt tekst, og utvetydiggjør hver enkelt entitet ved å ta hensyn til konteksten. Prosessen med å hente ut entiteter tas hånd om av et eksternt rammeverk, mens utvetydiggjøringsprosessen fokuserer på entitetslikheter, hvor likhet er definert etter hvor like entitetenes kategorier er, som vi finner ut ved å bruke informasjon fra en strukturert kunnskapsbase kalt DBpedia. Vi baserer denne tilnærmingen på antagelsen om at like entiteter vanligvis forekommer i nærheten av hverandre i tekst, dermed velger vi entiteter som virker lik andre entiteter i nærheten.

Eksperimenter viser at vår implementasjon ikke er like effektiv som noen av de eksisterende løsningene vi bruker som referanse, og at vår tilnærming har noen svakheter som bør bli adressert. DBpedia er for eksempel ikke en så konsekvent kunnskapsbase som vi skulle ønske. I tillegg feiler ofte rammeverket vi bruker for å hente ut entiteter fra teksten med å reprodusere samme entitetsett som datasettet vi bruker for evaluering. Løsningen vår viser likevel lovende resultater i mange situasjoner.

Acknowledgements

I would first like to thank my supervisor Associate Professor Heri Ramampiaro for his support throughout the work with this thesis. His help with forming the research questions, and guiding me with both the writing process and the general work flow has been invaluable to me during this work.

I would also like to thank my fellow students. Your presence have been to my help both academically at the office and socially outside the office. You have not just been of help during the last couple of months, but during my whole study the last couple of years.

My friends outside study also deserve a thank, as you have been of large help when I have needed some relaxation from the research. Especially, I want to thank Catrine Emilie Jensen and Olav Mogstad for reading through the thesis and correcting mistakes.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

People are sharing more and more information over the Internet, where a large amount of this information is unstructured textual information written by and for people. In some cases, the reader may not have the necessary background knowledge to fully comprehend the meaning of an online article, and must therefore make an effort on their own to obtain this knowledge, e.g., by using a Web search engine, to truly benefit from the reading. Another scenario may be that the reader wants more information about the subject they have just read about, and is again led to find more information on their own.

It would generally be beneficial if Web sites could assist the readers who want more information. Simply writing more information for each article is not optimal, since it may lead to impractical article lengths. Additionally, it is a fair assumption that readers have a varying need of extra information. Simply adding a lot of information on every article is therefore not optimal. A better solution is to provide links to other articles with more detailed information about certain topics that is known to be relevant for the actual article, which simplifies the information gathering process for the user.

Consider a news article on a Web page of a newspaper about a blooming conflict between two nations in the Middle East. A reader who is normally not that interested in geography, or general information about the Middle East, still finds the situation intriguing and wants to learn more about the two nations. A way to satisfy this need from the newspaper's point of view could be to provide links to an article or a Web site with more information of the two nations, making it easier for the reader to acquire more information.

Most newspapers today are aware of this, so the journalists writing the articles usually supply hyperlinks to previous articles published on their Web site about more or less the same subjects. Newspapers might have an extra interest to keep the reader on their platform, so links to external sources are usually only used to reference their original source for the article.

Consider an article about a recent football match. A reader might want to read more about one of the teams. However, since newspapers normally do not have an article about every football team (or other subjects, for that matter) they mention, the reader is normally provided links to articles about previous matches played by the two teams, rather than general information about any of the two teams. Again, the reader is left to find the necessary resources on their own.

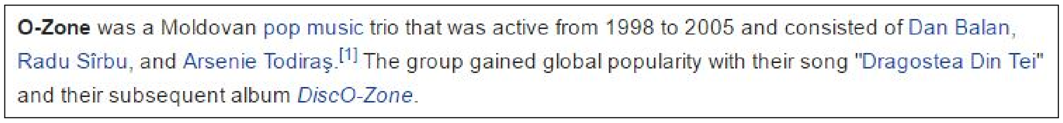

O-Zone was a Moldovan pop music trio that was active from 1998 to 2005 and consisted of Dan Balan, Radu Sîrbu, and Arsenie Todiraş.[1] The group gained global popularity with their song "Dragostea Din Tei" and their subsequent album *DiscO-Zone*.

Figure 1.1: Screenshot from Wikipedia, showing links to articles with a more detailed explanation of different entities.

One Web site that really emphasizes on using this strategy of linking entities in a text to another article with more information about that exact subject, is `wikipedia.org`, which is the worlds largest encyclopedia. This annotation is showed in Figure 1.1. The English version consist of approximately 5.250.000 articles of unique subjects.[1] With all this data they are able to reference to other articles to better explain the content in the actual article with clickable hyperlinks. This makes it easy for the reader to get more information about certain topics if that is desirable.

Wikipedia is community-driven, where every article is written by their almost 30 million registered users. With so many unique users, it is natural

[1]Statistics available at `https://en.wikipedia.org/wiki/Special:Statistics`

that they cover quite a lot of different topics, so that pretty much everything is covered, including music, sport, politics, geography, religion, history, literature, science, and so on.

All links in Wikipedia is manually added by the writers. A link is only added where the writer thought it would add value to the future readers. In the said figure, we see that *"pop music"* is linked to an article with more information about that topic, while *"1998"* and *"2005"* do not have any attached links, even though Wikipedia also contains dedicated articles for both these years, with births, deaths, and other events for those specific years. It is, however, up to the writer to determine if it is really necessary with a link in these cases.

Both newspapers and Wikipedia adds links to enhance the reading experience, along with several other services. Today, most of these services have in common that all links is added manually by the writer, as mentioned for both newspapers and Wikipedia. The task of automating this process is however an important research field in computer science. This is called ***entity linking***, and may be split into several smaller sub-problems, such as detecting actual entities in the text, solving ambiguous cases, and ranking possible candidates in a knowledge base to decide what best fits the entity.

If a system with a near-human precision could be designed for entity linking, it would mean that we could restructure all existing digital text to follow this design pattern, where every mentioned entity with a node in a knowledge base could get a direct link in the text. This may enhance the reading experience. Also, it would no longer be necessary for the writers at newspapers, Wikipedia, or other services, to add links manually, and they may therefore save time and resources.

This is only one use-case for such a system, but with this in mind we see that it would be beneficial to keep exploring possible techniques and technologies which may help us design such a system.

1.2 Problem Specification

With this basis our main research question will be:

RQ: **How can we implement an entity linking system that selects the most relevant candidate article, taking entity ambiguity into account?**

This research question can again be split into several sub-questions:

RQ1: **What kind of data and methods can be used to disambiguate ambiguous entities?**

RQ2: **How do today's entity linking systems select what candidate article should be used for each entity?**

RQ3: **What is the best method for selecting correct article for an entity, and how can we maximize the efficiency for such a system?**

1.3 Project Scope

We need to set a scope for this master thesis due to time and resource limitations. We will mainly focus on the entity disambiguation task, rather than the entity extraction task within entity linking. Since we have neither time or resources to build a platform that can fully understand unstructured written text, this is beyond the scope of this project, and we use a third party framework for entity extraction.

Additionally, we only consider input text in English, and ignore potential misspellings. We also exclude rich content such as video, audio, figures, images or tables, meaning we only consider raw text data as input.

1.4 Report structure

The remainder of this report is structured as follows:

Chapter 2: Introduction to different elements, such as research areas one should be familiar with when working with entity linking.

Chapter 3: Presents state-of-the-art techniques within entity linking by looking at several modern implementations and performing simple experiments on them.

Chapter 4: Explains the design and thinking behind our implementation, including a closer look at some especially interesting aspects.

Chapter 5: Describes the experiment design and results.

Chapter 6: Looks at the results from the experiments, and draws parallels to the implementation.

Chapter 7: Presents a conclusion for the research, and suggests potential future work.

Chapter 2

Background and Theory

2.1 Information Extraction

Information extraction (IE) is a field in computer science concentrated around the task of automatically extracting structured information from unstructured or semi-structured machine-readable documents. IE often involves processing natural human languages (such as text written by humans), often referred to as **natural language processing (NLP)**.

2.1.1 Named Entity Recognition

Named Entity Recognition (NER) represents the task of recognizing items, or entities, in an input text which maps to proper real world names. This might be people, places, happenings, companies, etc. In short, a NER-system takes unstructured text as input, and outputs all recognized entities in the text. The NER-problem is highly relevant in our case, since these named entities in the text usually will be appropriate for entity linking (see Section 2.1.3).

For earlier systems, the most popular approach has been to use handcrafted rule-based algorithms, often using specific features [32]. Features are descriptors or characteristic attributes of words designed for algorithmic consumption. This could for instance be a Boolean value representing whether a word is capitalized or not, a list over defined named entities, or similar. This works well in simple cases, but sentences in the real world tend to be quite complex. Because of this, most of today's systems use supervised learning, where rules and entity lists are created from a large amount of annotated corpus.

NER is often extended to also involve classification of the found entities, telling if the given entity is a person (PER), organization (ORG), geopolitical entity (GPE), etc. This is called **Named Entity Recognition and Classification (NERC)**.

2.1.2 Named Entity Disambiguation

Named Entity Disambiguation (NED) represents the task of disambiguating entities within a text, or simply said, choosing the correct article describing an entity when multiple candidates appear to be relevant for the given entity. These candidates are usually found using a knowledge base, looking for articles matching the entity name.

Consider the following sentence as an example: *"Mickey Mouse has a pet dog called Pluto"*. *Mickey Mouse* is relatively straightforward, and should point to the Disney-character. For humans, it is also quite straightforward to see that *Pluto* should point to the Disney-character, which is in fact Mickey Mouse's pet dog. For computers however, it is not necessarily that easy. Both `google.com` and `wikipedia.org` returns Pluto the dwarf planet as the top hit when searching for "Pluto". This might lead a computer to think that the entity mention represents the dwarf planet rather than the Disney-character. In circumstances like these we need a method to resolve the conflicts of multiple hits in the knowledge base, using some kind of ranking algorithm to pick the correct candidate.

Ranking the possible nodes is not necessary a straightforward task either, since these mentions might be highly ambiguous, making several nodes a possible match for the given entity. There are a number of different ways to find the best candidates. For instance, one can use some kind of machine learning approach such as Naive Bayes or decision trees, or one can check similarity of each candidate up against other elements of the actual text (context-wise). We will get back to this as we take a closer look at different systems later (see Chapter 3).

2.1.3 Entity Linking

Entity linking is the general all-inclusive task for linking entities in an input text to the correct node of a knowledge base [35]. To achieve this, one must start by finding named entities with a NER-analyzer. After finding all entities in the text, we must find the node in our selected knowledge base that best fits that exact entity using a NED-system, before we produce the actual link.

Given the input text *"Gerrard used to play alongside Carragher for Liverpool"*, three entities should be extracted: *Gerrard*, *Carragher* and *Liverpool*. These three entities should then be identified and point to the two ex-footballers Steven Gerrard and Jamie Carragher, and the football club Liverpool F.C. These entities should be highlighted, with a link to, e.g., Wikipedia.

If an entity linking system fails to resolve ambiguous entities, it is prone to frustrate the reader more than helping him or her, since the provided link might mislead the reader to an article that is actually unrelated to what is expected. In our mentioned example, the system might incorrectly identify *Liverpool* to represent the city, rather than the football club. Therefore, it is very important to have well-working mechanisms to handle ambiguous entities.

2.2 Semantic Web

The semantic web is an extension of the Web through standards by the World Wide Web Consortium (W3C), making content that is meaningful to computers [2]. Traditionally, the Web's content has been designed for humans to read, not for computer programs to process. The semantic web aims to bring structure to the meaningful content of Web pages, allowing software agents to automatically carry out tasks for users.

The use of hashtags (#) on social medias like Facebook and Twitter is one simple example of the semantic web in practice [12]. The users tag their posts with relatively simple words or phrases without whitespace, e.g., one user may take a picture of a sunrise and add *"#sunrise"* to the post. This enables Web programs to retrieve posts probable to contain a sunrise in a simple manner.

People are moving towards building a semantic web where relations can be established among any online piece of information. The data language, called Resource Description Framework (RDF), names each item and their relations in a way that allows computers to automatically interchange the information. Structured knowledge bases, which we will get back to shortly, are important contributors to the semantic web.

The entity linking task is dependent on the semantic web, as we need to discover relations between different entities in order to disambiguate them. By performing entity linking, we can also obtain a lot of structured information about entities in the text, which is really the core idea for the semantic web.

2.3 Knowledge Base

The Oxford Dictionaries defines a knowledge base (KB) as:

1. *a store of information or data that is available to draw on*
2. *the underlying set of facts, assumptions, and rules which a computer system has available to solve a problem*[1]

Thus, according to the Oxford Dictionaries, a KB can be used to store structured and unstructured data, and make it available for a computer system. This really comes in handy for entity linking, as we need to both provide and retrieve information about entities. An entity linking system aims to supply links to a KB, and may also use information stored about the entity candidates in the KB in order to disambiguate entities.

Even though it would be theoretically possible to obtain information about entities by parsing raw text from an unstructured KB, this is infeasible as one would need an incredibly complex parser to get all the necessary information for all different kinds of entities. Therefore, it would be a great advantage if the KB has some kind of mechanisms to keep track of relations

[1]`https://en.oxforddictionaries.com/definition/knowledge_base`

between entities. These relations might be checked up against the context where the entity appears in the text, and thus help us find the best entity candidate.

2.3.1 Wikipedia

Wikipedia[2] might be the best known KB today, as it is the worlds largest encyclopedia. Wikipedia is community-driven, where the users write on a voluntarily basis and is allowed to edit (almost) any article at any time. Since anyone can sign up, some users will enter incorrect information, either on accident or on purpose, which may lead to the user being banned from further writing. However, with such a large user base as Wikipedia has, wrongly entered information usually gets fixed rather quickly.

When Wikipedia is used as KB for entity linking the process is often called wikification. The data on Wikipedia is semi-structured, since the topics of each page is known and relationship between the different topics is also maintained. Infoboxes, tables, lists, and categorization data is considered as structured data in Wikipedia articles. However, it is not fully understandable for a computer exactly what the text for each article describes.

The English version of Wikipedia consist of over 5M articles of different subjects, created by almost 30M users (with writing permission). With such a wide user base, it is fair to assume that the data is quite up-to-date at any time. This is also reflected by the statistics, saying they have almost 3,5M edits per month, which corresponds to around 80 edits per minute.[3] Wikipedia has articles written in almost 300 different languages, but the amount of articles and users for each language naturally varies. For instance, *Norwegian (Bokmål)* consist of approximately 450.000 articles written by 370.000 users.

[2]`www.wikipedia.org/`

[3]`https://stats.wikimedia.org/EN/SummaryEN.htm`

2.3.2 DBpedia

DBpedia[4] extracts structured, multilingual knowledge from Wikipedia and make this available to its users using semantic web (see Section 2.2) and Linked Data technologies [24]. With its strong link to Wikipedia, it also supports many languages and the information evolves as Wikipedia changes. Because of the structured information in DBpedia it is also possible to make more complex queries using SPARQL, e.g., *”Give me all cities in Norway with more than 40.000 inhabitants”* or *”Give me all nations who participated in the FIFA World Cup 2014”*. Reminding more of a SQL query than a plain word search supported by Wikipedia.

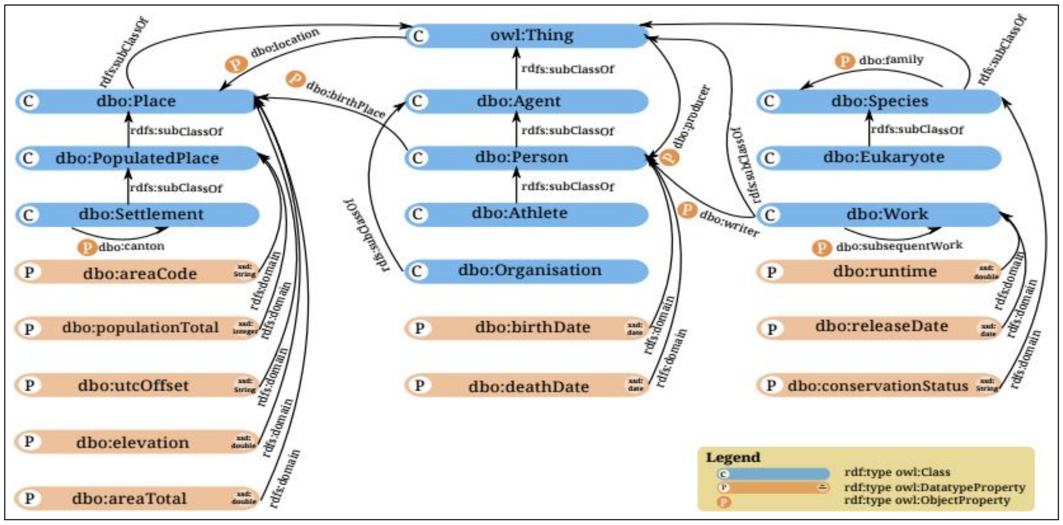


Figure 2.1: Snapshot of a small part of the DBpedia ontology.

DBpedia extracts the structured information from Wikipedia, such as infobox templates, categorization information, images, geo-coordinates, links to external Web pages, disambiguation pages, redirects between pages, and links across different language editions. This information is then turned into a rich knowledge base. The generated DBpedia ontology consist of 320 classes (person, organization, place, species, etc.), which is described by 1.650 different properties (birth date, area code, release date, etc.).[5] A simple overview is shown in Figure 2.1.

[4] `wiki.dbpedia.org/`
[5] `http://mappings.dbpedia.org/server/ontology/classes/`

DBpedia has a new release approximately twice every year, but this may vary. These releases typically have some updates related to the used ontology model, in addition to a "fresh" and updated dump of Wikipedia data. DBpedia Live is a service which ensures a continuous synchronization between DBpedia and Wikipedia, with a small delay of at most a few minutes [31].

2.3.3 Wikidata

Wikidata[6] by the Wikimedia Foundation aims to create a free KB that can be read and edited by both humans and computers, meaning they must structure the data making it as easy as possible for computers to understand [41, 11]. This project intends to structure all the data of Wikimedia sister projects as Wikipedia and others. Wikidata is an ongoing project and is still under active development. Data can be retrieved by requesting their official API.

Data stored in the Wikidata KB are called items and can have labels, descriptions and aliases in all languages. Wikidata provides a set of statements about something, rather than stating single truths. For instance, instead of stating that Berlin has a population of 3.5M, Wikidata contains the statement about Berlin's population being 3.5M as of 2011 according to German statistical office. This way, Wikidata can offer a variety of statements from different sources and dates. These statements may again be ranked to define their status (preferred, normal or deprecated).

2.3.4 Freebase

Freebase[7] was a scalable tuple database used to structure human knowledge powered by Google, which was thought of to be "Wikipedia for structured data" [5, 4]. Freebase was collaboratively created, structured and maintained, before it was officially shut down on August 31, 2016. Their last dataset is still available, but they will not supply any updates in the future.

[6]`www.wikidata.org`
[7]`https://developers.google.com/freebase/`

Google discontinued working on the Freebase service as a standalone project since the Wikidata project (see Section 2.3.3) was a fast growing project with an active community and with the same objective as Freebase[8]. Therefore, the Freebase team decided to help transfer the data in Freebase to Wikidata, rather than keep competing with them.

2.3.5 YAGO

YAGO (Yet Another Great Ontology)[9] builds on entities and relations, which is automatically extracted from Wikipedia [38]. This extraction step utilizes the fact that Wikipedia has category pages offering general details about the entities. The YAGO model is able to express entities, facts, relations between facts and properties of relations. YAGO enables users to browse their KB with a graph browser, ontology browser, in addition to an available SPARQL endpoint, all available within their demo page.

The latest version of YAGO, *YAGO3*, combines the information from Wikipedia in multiple languages [25]. The multilingual information is fused with the English version in order to build one coherent knowledge base. To achieve this, they make use of the categories, the infoboxes, and Wikidata, and learn the meaning of infobox attributes across languages. By doing this they aim to represent multilingual entities in one consistent way.

2.4 Evaluation Methods

In order to evaluate a system's performance, there is a need for an evaluation method with corresponding scoring metrics. Some project teams implementing entity linking systems decide to also invent and develop their own evaluation method and data sets. Their possibility to design an evaluation method and data sets to fit their systems actual performance perfectly, rather than making an general and fair evaluation is a potential problem. This might make comparisons of different systems using their own evaluation unfair, since they might get an undeservedly good score for their evaluation, while

[8]https://plus.google.com/109936836907132434202/posts/bu3z2wVqcQc

[9]http://www.mpi-inf.mpg.de/yago

performing considerably worse for other evaluation methods and data sets that might be more close to the real world.

Workshops or conferences where multiple project teams submit their systems according to a common problem description are typically also evaluated using the same methods and metrics. This is naturally the fairest way to compare systems, since all systems have the same goal of what they want to achieve, and on this basis they are put on the same tests to measure their actual performance. These tests usually focus more on effectiveness, rather than efficiency.

TAC-KBP is one example of such conference, which we will also come back to in more detail later in this report. Their evaluation method calculates precision and recall between gold (G), which is manually annotated, and a system's (S) annotations [22]. The annotations are a set of distinct tuples. Values for precision (P) and recall (R) are combined as their balanced harmonic mean (F_1), which is used to compare each system:

$$P = \frac{|G \cap S|}{|S|} \qquad R = \frac{|G \cap S|}{|G|} \qquad F_1 = \frac{2PR}{P + R}$$

The TAC-KBP evaluation tool is open source and freely available for download[10], making it easy to use for testing during the development process as well. The source collection includes 90,000 documents, where 500 documents are selected for evaluation and manually annotated[11].

The precision, recall and balanced harmonic mean scores are present in most evaluations, without any large deviation. What varies more, however, is how the documents used for evaluation are selected. Since different systems typically perform unequal on the same data sets, one might be tempted to select documents that are well fitted for your system's strengths, as earlier discussed.

There are a number of freely available datasets where the text is already annotated, and each entity is marked with the ID of the article it should point to. Considering the method for calculating precision, recall and F measure should be similar for most systems, it should not be a big problem to compare different systems using the same dataset.

[10]`https://github.com/wikilinks/neleval`

[11]`http://nlp.cs.rpi.edu/kbp/2016/taskspec.pdf`

Chapter 3

State of the Art

3.1 Related Work

A lot of work has been conducted towards entity linking the recent years, which has resulted in several different solutions. Different approaches have been tried, some with success, others less so. *Wikify!* did in many ways make the entity linking task ”popular” with their research, leading to Knowledge Base Population (KBP) being implemented as a track of the annual Text Analysis Conference (TAC). It is fair to assume that all serious entity linking systems have taken some inspiration from these two sources.

3.1.1 Wikify!

Wikify! is an early entity linking system from 2007 developed at the University of North Texas [27]. This system identifies important concepts in any given input text (keyword extraction) and automatically link these concepts to the corresponding Wikipedia pages (word sense disambiguation).

Wikify! uses a set of unsupervised keyword extraction techniques. The problem of finding entities is done by assuming to be working under a controlled vocabulary setting, where all keywords in the vocabulary are accepted phrases. This way, nonsense phrases like, e.g., *”products are”* are ignored.

The unsupervised keyword extraction algorithm works in two steps: candidate extraction and keyword ranking (see Figure 3.1). The *candidate extraction* step parses the input and extracts all entities that are also present in the controlled vocabulary. The *ranking* step assigns a numeric value to each candidate, based on the likelihood that a given candidate is a valuable keyphrase. In order to decide how many links should actually be added, a simple statistical analysis of Wikipedia was performed, finding that approximately 6% of all words is annotated, which is also the default ratio used for Wikify!.

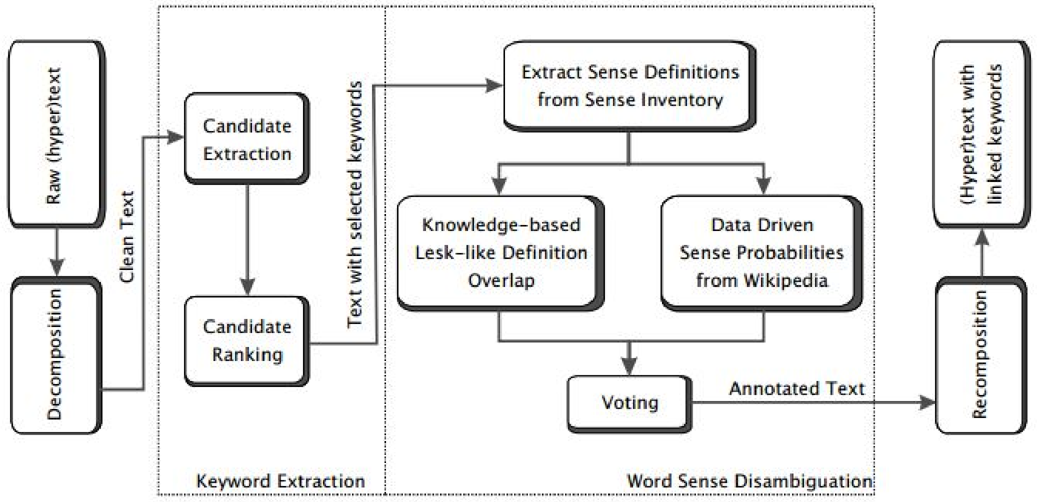

Raw (hyper)text
Decomposition
Clean Text
Candidate Extraction
Candidate Ranking
Text with selected keywords
Keyword Extraction
Extract Sense Definitions from Sense Inventory
Knowledge-based Lesk-like Definition Overlap
Data Driven Sense Probabilities from Wikipedia
Voting
Annotated Text
Word Sense Disambiguation
Recomposition
(Hyper)text with linked keywords

Figure 3.1: Architecture of the Wikify! system.

Two different disambiguation algorithms were tried out. The first, a knowledge-based approach, attempted to identify the most likely meaning for a word in the actual context (current paragraph) based on a measure of contextual overlap between the dictionary definitions of the ambiguous word. The second approach was a data-driven method using a Naive Bayes classifier to find the best candidate.

The data-driven approach generally outperformed the knowledge-based approach, but a combination of the two gave the least amount of incorrect annotations, but it also missed out on some words it should have annotated.

The system was tested by 20 users with mixed backgrounds, each evaluating 10 documents and trying to determine if each document was annotated manually by a human, or by a computer (using Wikify!). The human version was correctly identified in 114 of the 200 cases, which gives an accuracy of 57%. This is close to the ideal accuracy, which is 50%, meaning the human and computer would have been indistinguishable.

Not long after the research of the Wikify! team was published, another study at the University of Waikato with some improvements of the Wikify! system was published [28]. The central difference between this and the original Wikify! system is that this solution finds entities in the text by using disambiguation to inform detection. A machine learning approach, using the actual links within Wikipedia articles is used as training data. The C4.5 algorithm is used to generate a decision tree. This newer system outperforms the original system when it comes to detecting entities within the text, or keyword extraction.

3.1.2 Knowledge Base Population

The goal of Knowledge Base Population (KBP) is to develop and evaluate technologies for populating knowledge bases from unstructured text. The Text Analysis Conference (TAC) has a series of workshops organized to encourage research in Natural Language Processing (NLP)[1]. In recent time, much focus has been put towards KBP[2], currently consisting of five tracks: Cold Start KBP, Validation/Ensembling Track, Event Track, Belief/Sentiment Track, and Entity Discovery and Linking.

Cold Start KBP Aims to build a KB from scratch using a given document collection and a predefined schema for the entities and relations that will form the KB. In addition to an end-to-end KB Construction task, the track also include a Slot Filling task to fill in values for predefined attributes for a given entity.

Validation/Ensembling Track Focuses on the refinement of output from slot filling systems by combining information from multiple slot filling systems, or applying more intensive linguistic processing to validate individual candidate slot fillers.

[1] `http://tac.nist.gov/`

[2] `https://tac.nist.gov/2016/KBP/`

Event Track Aims to extract information about events such that the information would be suitable as input to a KB. The track includes Event Nugget tasks to detect and link events, and Event Argument tasks to extract event arguments and link arguments that belong to the same event.

Belief/Sentiment Track Aims to detect belief and sentiment of an entity towards another entity, relation, or event.

Entity Discovery and Linking (EDL) The most relevant track in our case, as it aims to extract entity mentions from a source collection of textual documents in multiple languages (English, Chinese, and Spanish), and link them to an existing KB. A system is also required to cluster mentions for those entities that do not have any corresponding KB entries.

KBP has been an active track of TAC every year since 2009, where the results and work for each conference builds on the work and experiences from previous years. In the latest conference with published results, from 2015 (building on the work from 2014 [21]), the focus of the EDL track was towards *Tri-lingual Entity Discovery and Linking* [22]. I.e., given input text in English, Spanish and Chinese, all entity mentions should be discovered and linked to an English knowledge base. The 2016 conference is completed, but its results are yet to be published at the time of writing.

The participants' annotators were guided to use the context in order to confidently identify the intended referent of any entity mention within the text [10]. First, the annotators would have to find the entities and indicate their type (person, organization, etc.), before linking to a node in the KB.

Supervised learning usually produces better results than unsupervised learning, and state-of-the-art entity discovery and linking methods rely on entity profiling and collective inference. For high-resource languages like English, it is possible to use some advanced knowledge representations to effectively select semantic neighbors for entity profiling and collaborators for collective inference.

Nominal mentions[3] is still a problem, especially when it comes to separating specific and generic mentions. However, they can often be resolved by looking

[3]Entity mention that is not composed solely of a named entity or pronoun

for certain keywords in the sentences, but in some cases there are also a need for some background knowledge. When it comes to linking a nominal mention to the KB, the most effective approach is to apply within-document coreference resolution to resolve it to a name mention.

Several other problems are still present within entity linking as well. In Chinese, e.g., geographical and political names are sometimes abbreviated as single characters, which is highly ambiguous in various contexts. In these cases, the entity linking system was better off by assigning more weight to context similarity rather than popularity. Also, some instances get a wrong link due to a too weak knowledge representation, as the connection between instances in the same sentence was not discovered.

We will take a closer look at the top three entity linking systems submitted by the participating groups: RPI, IBM and HITS.

RPI

The RPI system extract English name mentions by using a trained linear-chain conditional random field model, while the Stanford name tagger is used for Chinese and Spanish [20]. They also encode several regular expression based rules to extract poster name mentions in discussion forum posts. Person nominal mentions are detected by looking for indefinite article (e.g., *a/an*) and conditional conjunctions (e.g., *if*).

Their entity linking system is domain and language independent. It is based on an unsupervised collective inference approach. Given a set of English entity mentions, they first construct a graph for all entity mentions based on their co-occurrence within a paragraph. Then, all entity mentions are assigned a list of entity candidates, with a computed importance score by an entropy based approach.

Finally, the system computes similarity scores for each entity mention and candidate pair, and selects the candidate with highest score. For Chinese or Spanish input rather than English, the mentions are first translated into English using name translation dictionaries.

IBM

The IBM mention detection system was based on a combination of deep neural networks and conditional random fields, depending on the input language [36]. Neural network was used for English and Spanish, while conditional random field models was used for Chinese.

The entity linking system is based on a language independent probabilistic disambiguation model. They partition the full set of entity mentions of an input document into smaller sets of mentions which appear near one another. For deciding resources to the entity mentions, they use a maximum entropy model, resulting in probability values for each candidate.

Several feature functions are used in order to obtain the correct resource. *Local features* include counting the number of mentions whose surface form matches exactly with one of the names for the linked entity stored in Wikipedia. Additionally, they have Boolean values representing if all mentions match a Wikipedia article exactly, and if the mention's surface form is an acronym for a name of the linked entity in Wikipedia, among others.

Some of the *global features* include making use of Wikipedia's category information to find patterns of entities that commonly appear next to each other. In order to get better results, they remove common Wikipedia categories which are associated with most entities, like *"Living People"* etc., since they have lower discriminating power.

HITS

The core of HITS' entity linking system is a pipeline of sieves that perform the subtasks necessary for entity linking [18]. They first perform a high-precision mention detection and disambiguation, used as seeds for iterative application of subsequent sieves. This result is then used to consider the context also in the mention detection step.

The first couple on sieves focus on identifying, almost, unambiguous mentions, which are then verified through information in the knowledge base. In cases that are somewhat ambiguous, they are omitted, and evaluated again later considering the context found in earlier sieves.

Their disambiguation process fix one argument and a relation type, and then look for a matching second argument in the input document. More concretely, they compile a set of paths in the knowledge base graph that connects entity types of interest. Following these paths for each linked mention of matching entity type, they query related entities in the knowledge base, and then check if known surface forms of any of the related entities occur in the text.

After global disambiguation, they apply several post-processing heuristics that are mainly designed to increase linking recall. For instance, all uncertain cases are assigned to the most frequent sense according to Wikipedia article links.

3.1.3 Knowledge Base Acceleration

The goal of Knowledge Base Acceleration (KBA) is to help humans expand knowledge bases by automatically recommending edits based on incoming content streams. The Text REtrieval Conference (TREC)[4] has a series of tracks wanting to encourage research in information retrieval from large text collections. KBA was one such track[5], which aimed to develop techniques to dramatically improve the efficiency of (human) knowledge base curators by having the system suggest modifications or extensions to a knowledge base based on its monitoring of the data streams.

The KBA track was running three successive years, from 2012 to 2014, before it was succeeded by the Dynamic Domain track in 2015, which builds on the foundations of KBA. KBA has strong connections to entity linking, as we need to find the correct node we want to update in the knowledge base. In the case of KBA, one must also consider if this new information is relevant, i.e., if this new piece of information should be stored in the KB at all, also called Vital Filtering [16].

Many large knowledge bases, such as Wikipedia, are maintained by small workforces of humans who cannot manually monitor all relevant content streams. This lead to most entities lagging behind current events. KBA aims to minimize this gap, by filtering streams of text for new information about entities, and suggest relevant updates.

[4]`http://trec.nist.gov/`

[5]`http://trec-kba.org/`

Vital filtering can be viewed on as a binary classification problem. However, computing a potentially large set of features for every single document-entity pair is not feasible, thus a more efficient solution is required. One approach to solve this is to implement multiple binary classification steps to decide whether a document is central/vital or not [1].

When using a 2-step approach, each document is classified as either central, or not central. Each of these document-entity pairs are assigned a score in the (0, 1000] range, where a higher score will indicate that the document is more central. The non-central documents are mapped to the (0, 500] range, while the central documents are mapped to the (500, 1000] range.

When using a 3-step approach, the documents are first classified as relevant (range (500, 1000]), or not relevant (range (0, 500]). However, the documents classified as relevant are classified once again with more precision, as either central (range (750, 1000]), or not central (range 500, 750]). We take a closer look at three of the systems submitted at the conference in 2014.

PRIS

The PRIS system uses DBpedia as external source data to do query expansion and generates directional documents to calculate similarities with candidate worth citing documents for the Vital Filtering task [34]. The system then utilizes a pattern learning method to do relation extraction and slot filling.

KMG

The KMG system's strategy for vital filtering is to first retrieve as many relevant documents as possible and then apply classification and ranking methods to differentiate vital documents from non-vital documents [23]. They first index the corpus and retrieve candidate documents by combining entity names and their redirect names as phrase queries. The system then learn to rank documents by leveraging four types of features: time range, temporal feature, title/profession feature, and action pattern.

Distributed Non-Parametric Represention

One system focused on Distributed Non-Parametric Represention for vital filtering [6]. They introduce a word embedding-based non-parametric representation of entities. The word embeddings provide accurate and compact summaries of observed entity contexts, further described by topic clusters that are estimated in a non-parametric manner. Additionally, they associate a staleness measure with each entity and topic cluster, dynamically estimating their temporal relevance.

3.2 Related Technology

There are several entity linking systems available today. So when we wanted to take a closer look at some of them, we first had to decide which one to use and which to omit. To get a relatively fair comparison of the systems, we will use the following set of criteria for each of them:

Free to use The task of entity linking is a relatively hot research topic, so finding freely available services on the Web was not a big problem. Commercial actors typically want to keep their knowledge to the domain more to themselves, and will not necessarily give a thorough enough description of their approach, making it hard to learn of their work (see Documentation below).

Documentation In order to get an understanding of the systems, we need some kind of documentation of them. Preferably, we want some documentation of each of the API endpoints, in addition to a more detailed description of which methods and techniques is used to extract the entities, and how the best candidate article for each entity is chosen. This could, e.g., be in the form of a published scientific paper or similar.

There is no matter of course that all entity linking services is evenly well documented, or even has any documentation at all. In order to get a good enough insight to how each of the systems handles the requests and learn from their approach, we will only consider services with a sufficient documentation, considering both quality and quantity. However, we must expect that the services we have available has not prioritized documentation equally well.

English support To compare the services up against each other we want to use the same test text, which should be written in English. Many systems support multiple languages, but we only care about the support for English, which is also the most commonly supported language.

GUI application We wanted the ability to use the application through a GUI[6] because of its simplicity when testing it. A GUI for this kind of application typically consist of a field where the user may write or paste the input text to be processed, and an output field with the same text, but where the entities have a link attached to them. This makes it easy to check if the links makes sense, and to compare the outcome of each of the systems up against each other.

Programmatically available API We also want to see what kind of data is returned when making a call to the API[7], in order to get a better understanding of how the request is handled, and to see what data each of the services consider as relevant for the outcome.

Established Preferably we want systems that is established, respected and has a solid user base. This would indicate a relatively reliable, effective and efficient system. Of course, we must also consider newer systems which is not yet necessarily considered an established actor, but still serves a solid service. If the work of a project is cited in published scientific papers, that would indicate that their work and research is acknowledged in the scientific environment as well.

3.2.1 Existing Systems

On the basis of the points mentioned, we selected the following entity linking systems:

- TagMe
- DBpedia Spotlight
- AIDA

[6]Graphical User Interface
[7]Application Programming Interface

- AGDISTIS
- Babelfy
- Targeted Hypernym Discovery

There is no guarantee that these are indeed the best available systems for our purpose, but all of them complements the requirements we have set, and will therefore be more thoroughly studied.

Additionally, all systems will be put through a simple test, by programmatically sending a request to their API to annotate the following text:

"Gerrard used to play alongside Carragher for Liverpool"

This test is not by any means meant to measure the actual effectiveness of the systems, which would be unfair since some are built for considerably longer texts. The purpose of this test is to see what kind of additional data is returned for each of the systems. However, it will still be interesting to see if the annotators manage to map *"Gerrard"* to Steven Gerrard, *"Carragher"* to Jamie Carragher, and *"Liverpool"* to Liverpool F.C., rather than Liverpool the city.

3.2.2 TagMe

TagMe[8] is one of the most popular and well known entity linking services on the Web. The first version of TagMe was released in 2010 [14, 13]. The system was originally designed to process and annotate very short texts, consisting of few tens of terms on-the-fly, using Wikipedia as knowledge base endpoint.

Architecture

They index some useful information drawn from Wikipedia, e.g., anchors, which is the text in an article used as a link to another article. All anchors are stored in an *anchor directory*, unless it consist of only one character or just numbers, or if the anchor is especially rare. The final dictionary consists

[8]`https://tagme.d4science.org/tagme/`

of approximately 3M anchors. There is also created a *page catalog*, which consist of every (English) Wikipedia page, except disambiguation pages, list pages and redirection pages. This catalog consists of 2,7M pages. Finally, it is created an *in-link graph*, a directed graph whose vertices are the pages in the page catalog, and whose edges are the links among these pages. It is worth noting that this data was originally drawn from a snapshot of Wikipedia as of November 2009, and it is unclear whether it has ever been updated or not.

TagMe annotates a text via three steps: (anchor) parsing, disambiguation and pruning (see Figure 3.2). The anchors in the text are detected with parsing, by searching for multi-word sequences (up to 6 words) in the anchor dictionary. Disambiguation cross-reference each of these anchors with one relevant sense from the page catalog. Pruning may discard some of these annotations if they are not considered meaningful, using a scoring function that takes the link probability of the anchor and the coherence of its candidate annotation into account.

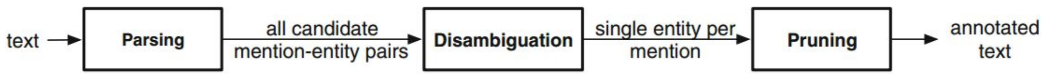


Figure 3.2: Annotation pipeline in the TagMe system [17].

On short texts TagMe's best disambiguator got an F-measure[9] of 91.2%, beating their predecessor Wikify! (see Chapter 3.1.1) who scored 88.3% using the same data set. Their best annotator also performed better than the one of Wikify! (F-measure of about 78 over 69). On longer texts, the performance of TagMe dropped down to a F-measure of about 72%, but was still surprisingly competitive, as Wikify! reached approximately 74%.

According to their Web site, their system has also been improved since the release: *"On August 2012, we have introduced major enhancements to the annotation engine and new services have been made available. This improved flexibility, precision and speed of TagMe (. . .)"*. However, they do not state any technical details about these enhancements.

[9]Measure of a test's accuracy, considering its precision and recall

Web Service

TagMe serves a RESTful API, which requires a "Service Authorization Token" (free to obtain) along with the annotation request. They have three different endpoints: one for annotating the text, one for just finding entities in the text, and one for calculating how semantically similar two entities are. All endpoints support multiple parameters, both required and optional, to fine-tune the results.

In our test, TagMe annotated all three entities correctly. Liverpool F.C. has an attached *"link_probability"* and *"rho"*, which estimates the "goodness" of the annotation with respect to the other entities of the input text, of 64% and 54%, respectively. Steven Gerrard and Jamie Carragher got around 5% and 27%, respectively. The former Swedish band *"Play"* was also suggested with a probability of 0,8% and rho equal to 0,4% for the same word as their name. The entities' *"dbpedia_categories"*, e.g., *"1980 births"* and *"Living people"*, were also returned, in addition to a short abstract of the given article.

3.2.3 DBpedia Spotlight

DBPedia Spotlight[10] is an open source project which started in 2010 for text annotation, naturally using DBpedia as knowledge base. The DBpedia Spotlight distribution also includes a jQuery plugin[11], in addition to a Java/Scala API (which can also be downloaded and run locally), and a demo page for testing.

Architecture

DBpedia Spotlight's approach works in four stages (see Figure 3.3) [26]. The *spotting* stage recognizes the phrases that may be a DBpedia resource in a sentence. This is performed using a string matching algorithm, looking for longest case-insensitive match. Since one often want to ignore common words, it is possible to set a flag to ignore spots that are only composed of verbs, adjectives and prepositions.

[10] `http://spotlight.dbpedia.org/`

[11] `https://dbpedia-spotlight.github.io/demo/dbpedia-spotlight-0.3.js`

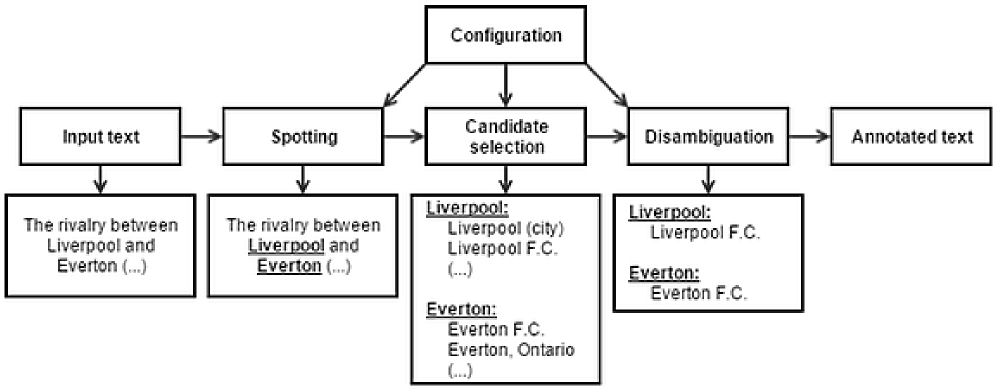

Configuration
Input text
Spotting
Candidate selection
Disambiguation
Annotated text
The rivalry between Liverpool and Everton (...)
The rivalry between Liverpool and Everton (...)
Liverpool: Liverpool (city) Liverpool F.C. (...)
Everton: Everton F.C. Everton, Ontario (...)
Liverpool: Liverpool F.C.
Everton: Everton F.C.

Figure 3.3: Annotation process for DBpedia Spotlight.

Candidate selection is performed afterwards, to map the spotted phrase to resources that are candidate disambiguations for that phrase. This is done using the DBpedia Lexicalization data set.

The *disambiguation* stage uses the context around the spotted phrase to find the best choice of the candidates. The candidates are ranked according to the similarity score between their context vectors and the context surrounding the surface form, using cosine as the similarity measure.

Configuration parameters can be used to customize the process for specific needs. This involves annotating only certain types, minimum number of inlinks of candidates in order to be annotated, annotate only phrases that are considered relevant to the topic, deny annotating phrases where contextual ambiguity is high, or demand a certain degree of confidence according to both topic and context before we annotate.

DBpedia Spotlight has later gone through some improvements making the annotation process faster, more accurate and easier to configure [8]. The spotting phase was improved by first generating candidates for possible annotations, then selecting the best candidates and discarding candidates with a score lower than a given threshold. The disambiguation process emphasizes a generative probabilistic model, telling how probable each of the candidates are to be the best alternative. These new implementations, in addition to some other minor modifications, improved the system rather drastically according to their subsequent tests. The tests covered runtime performance, space requirements, phrase spotting and disambiguation evaluation, where the newer approach got better results in all cases.

Web Service

Their Web service supplies endpoints for spotting entities to annotate, disambiguating already spotted text (chooses identifier for each entity given the context), spotting and disambiguation in one (annotation), and finding a ranked list of candidates instead of deciding on one.

DBpedia Spotlight includes a *"percentageOfSecondRank"* telling how much the best entity "won" over the second best candidate (lower score is more superior), *"support"* telling how prominent the entity is (number of inlinks in Wikipedia), and a *"similarityScore"* measuring the entity's similarity to other entities in that context.

Both Liverpool F.C. and Steven Gerrard was annotated, both with a similarity score of close to 100% and a very low score for percentage of second rank. These entities also had some attached types, such as *"Person"*, *"SoccerPlayer"* and *"SoccerClub"*, to mention a few. The word *"play"* was connected to a resource named *"Play (activity)"*, with a similarity score of 91% and a low score for percentage of second rank. *"Carragher"* was not detected as an entity, even if we lowered the confidence parameter with the request. However, if we added his first name (*"Jamie"*) to the input text, it was correctly annotated.

3.2.4 AIDA

AIDA (Accurate Online Disambiguation of Named Entities)[12] is an open source entity linking system created by the Max Planck Institute for Informatics, the same institute is the creator of YAGO, a knowledge base quite similar to DBpedia. Thereby, AIDA naturally use YAGO as knowledge base, in addition to Wikipedia. The backend is written in Java, using a Postgres database.

[12] `http://www.mpi-inf.mpg.de/yago-naga/aida/`

When referring to their work, they show to a scientific paper published in 2011 [19], even though they have published several papers within the same field of study since then. Looking at their commit log at GitHub[13], it seems like most development has been towards bug fixes since the original release, so we may assume that the documentation is still valid.

Ambiverse[14] is a spin-off from the original AIDA project, which creates solutions based on the AIDA technology. They have a free plan for users using less than 1,000 API calls per month, but charge money for users who need more. The documentation for Ambiverse mostly points to the AIDA documentation, so it is probable that most recent research goes towards improving this commercial application, rather than the original AIDA project.

Architecture

The Stanford NER Tagger [15] is used to identify noun phrases that potentially denotes named entities. All these entities are mapped to their Wikipedia disambiguation page (via YAGO/DBpedia). For mapping a mention to one exact entity candidate, the context around the mention is considered by computing similarity measures between the mention and potential candidates, in addition to the general popularity of the candidates. The coherence between entities is calculated by counting the number of incoming links their Wikipedia article have in common.

These measures for popularity, similarity and coherence is used to construct a weighted, undirected graph with mentions and candidate entities as nodes, as shown in Figure 3.4. The mention-entity edges are weighted with similarity measures in combination with popularity measures. The entity-entity edges are weighted based on their Wikipedia-link overlap, or type distance, or similar.

Given a mention-entity graph, the goal is ideally to compute a dense subgraph containing all mention nodes, and one mention-entity edge for each mention, meaning all mentions is disambiguated. Solving such a problem is however NP-hard, so an approximation algorithm is used in order to, hopefully, find a good solution in linear time.

[13]`https://github.com/yago-naga/aida`

[14]`https://www.ambiverse.com/`

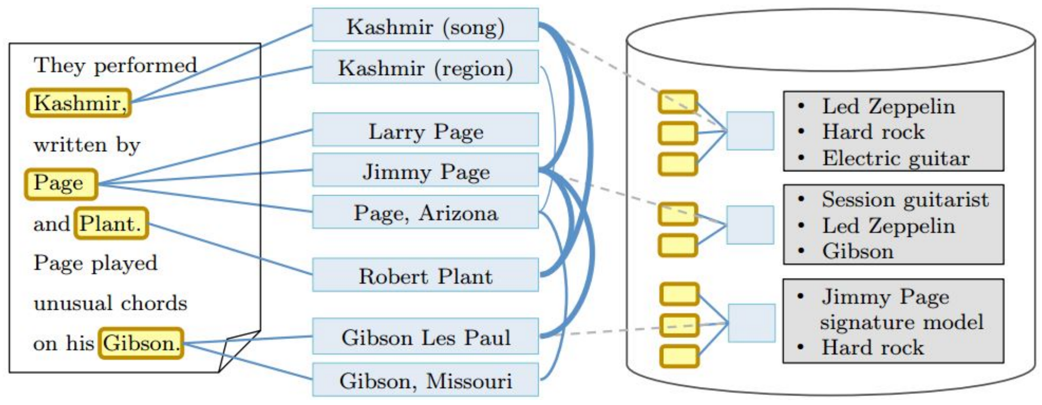


Figure 3.4: Mention-entity graph example, used by AIDA.

Web Service

The AIDA Web service consists of one single endpoint, with a required *text* parameter. This text is then annotated, and the result is returned in JSON format, with some additional information about the found entities, such as their position in the original text, length, disambiguation score, etc. Additionally, the endpoint accepts multiple optional parameters, that can be used to configure the disambiguation more specifically. These parameters include specifying which techniques and algorithms to use, specifying coherence threshold for annotation, and specifying which type of entities to annotate (e.g., annotate only person entities).

The AIDA Web service managed to annotate both *"Gerrard"* and *"Carragher"* correctly, both with an attached *"disambiguationScore"* of over 90%. *"Liverpool"* was not annotated unless we tuned the default settings, for instance by setting the *"tag_mode"* flag to manual and manually mark the entities in the input text with double brackets ([[*entity_mention*]]). With this approach *"Liverpool"* was also correctly annotated, with a disambiguation score of 34%. All entities were provided with a list of YAGO types connected to them, and a link to their respective Wikipedia page.

3.2.5 AGDISTIS

AGDISTIS (Agnostic Disambiguation of Named Entities Using Linked Open Data)[15] is another open source project concentrated around the disambiguation process released in 2014, hosted by the University in Leipzig [39, 40]. Their disambiguation framework is used by a named entity recognition framework called FOX[16], hosted by the same university, which forms a complete entity linking system when combined with AGDISTIS.

Architecture

Their approach consist of three main phases: retrieving all named entities from the input text, detect candidates for each of the detected named entities, and finally using the context to (hopefully) choose the optimal candidate (see Figure 3.5). All used algorithms have a polynomial time complexity, making AGDISTIS also being polynomial in time complexity.

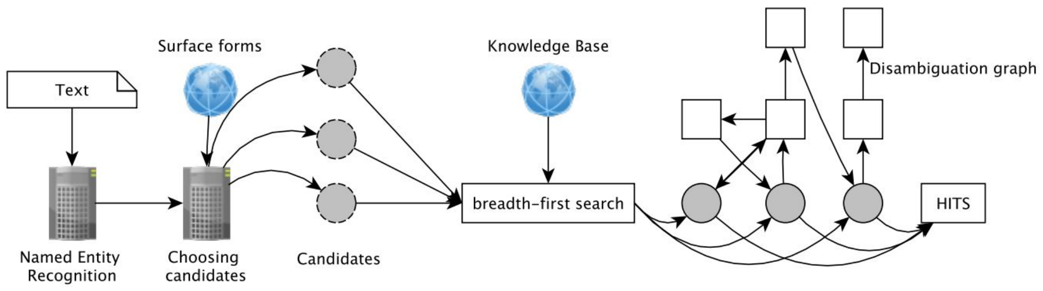


Figure 3.5: Overview of AGDISTIS.

All named entities are retrieved using a named entity recognition function (e.g., FOX [37]). After all entities are found, the search for candidate resources in the knowledge base begins. This can be done by using the entities' surface form, which are simply strings used on the Web to refer to given resources. For example, the surface form *"Washington"* can be used to refer to George Washington, Washington D.C., or Washington (U.S. state), to name a few. Several string normalization techniques is performed before searching for such surface forms, including eliminating plural and genitive forms.

[15]`http://agdistis.aksw.org/`
[16]`http://fox.aksw.org`

Given a set of candidate nodes, the computation of the optimal assignment is started by constructing a disambiguation graph. This graph is built in a breadth-first manner, and keeps track of resources and their resources. The HITS algorithm[17] is used on the disambiguation graph in order to identify the correct candidate node for a given named entity.

Web Service

Their RESTful service has one endpoint which annotates the given input text. Since AGDISTIS is only an entity disambiguation tool, it is worth noting that this system needs to be told where the entities are in the text. This is done by placing the entities within pre-defined entity tags. Using the FOX framework is an alternative, since it uses AGDISTIS for disambiguation, but also has built in NER tools to identify entities. Being an open source project, one can download the AGDISTIS system and run it locally as well.

To test this API, we had to format our test string by surrounding the entities with entity-tags, like this: *"⟨entity⟩Gerrard⟨/entity⟩"*. After this, the system was ready to annotate the text. *"Carragher"* was correctly annotated, while *"Liverpool"* was annotated to the city. Even more surprisingly, was that *"Gerrard"* was suggested to the Australian musician Lisa Gerrard. No additional information like confidence or similar measures was provided. We got the exact same annotations when testing against the FOX framework, but then we did not have to mark the entities manually.

3.2.6 Babelfy

Babelfy[18] is a graph-based approach to entity linking and word sense disambiguation written in Java, using BabelNet [33] as knowledge base. Their Web service can disambiguate any of the languages covered in BabelNet [30, 29].

[17]A graph-based link analysis algorithm that rates Web pages, not related to the system described in Section 3.1.2 with the same name

[18]`http://babelfy.org/`

Architecture

All sequences of words of maximum length five, which contains at least one noun and that are substrings of lexicalizations in BabelNet are identified, as they can potentially be linked to an entity in BabelNet. Because of this loose candidate identification, using substring matching instead of exact matching, we can also identify entities that is only partially written.

A semantic interpretation graph is created after identifying all potential entities. The set of nodes contains all potential candidate meanings of the found entities. Figure 3.6 shows one such graph.

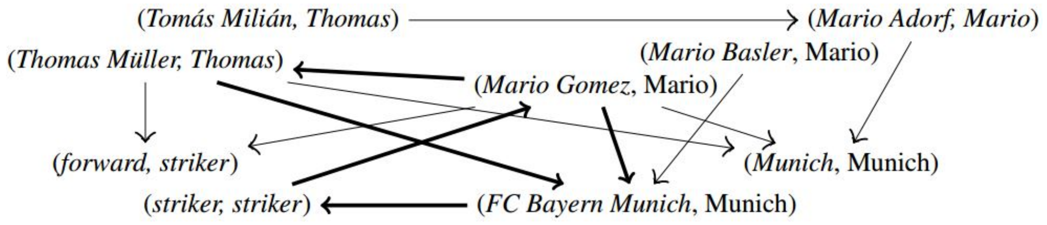


Figure 3.6: Semantic interpretation graph Babelfy builds for the sentence *"Thomas and Mario are strikers playing in Munich"*. The edges connecting the correct meanings are in bold.

A novel densest subgraph heuristic is used in order to reduce the degree of ambiguity while keeping the interpretation coherence as high as possible. The main idea here is that the most suitable meanings of each text fragment will belong to the densest area of the graph. The resulting subgraph will then contain those semantic interpretations that are most coherent to each other. The problem of identifying the densest subgraph of size at least k is NP-hard. Therefore, Babelfy uses a greedy 2-approximation algorithm, meaning it will return a subgraph of no more than twice the number of elements from the optimal solution in linear time.

This densest subgraph might still contain multiple interpretations for the same fragment, or even unambiguous fragments which are incorrect. Therefore, the final step is the selection of the most suitable candidate meaning for each fragment given a threshold value to discard semantically unrelated candidate meanings.

Web Service

In order to use the Babelfy API, one must first obtain an API key, which must be sent as a parameter with the requests. This key, the input text, and language are required for each request. Additionally, one can specify other various parameters, such as specifying which resource to use (WordNet, Wikipedia, or BabelNet), which entity types to annotate (named entities, word senses, or both), or if we want a scored list of candidates or only the top ranked, to mention a few.

When testing the API with our test sentence, *"Carragher"* was the only correctly annotated entity. *"Liverpool"* was linked to the city, rather than the football club. *"Play"* was annotated with a link to the BabelNet dictionary with a description of the word, while *"Gerrard"* was not detected at all.

Both *"Carragher"* and *"Liverpool"* got a *"coherenceScore"* of 1.0 and a *"globalScore"* of 0.5, while *"play"* surprisingly get 0.0 for both measures. However, we assume this is normal when a word is just linked to the BabelNet dictionary, and not an actual entity. If we use the full name for all entities, i.e., *"Steven Gerrard"*, *"Jamie Carragher"* and *"Liverpool F.C."*, all of them were correctly annotated.

3.2.7 Targeted Hypernym Discovery

Targeted Hypernym Discovery (THD)[19] performs classification of entities, and cross-link them to their representation in DBpedia [9]. The system is implemented in Java, and supports German and Dutch, in addition to English. As an end user, one may choose between multiple techniques and methods for both entity spotting and entity linking, in addition to some other specifications.

Architecture

THD's documentation is not quite as well-structured as the previously mentioned systems. The paper they refer to for citation [9] is quite shallow when

[19]`http://entityclassifier.eu/`

it comes to describing their approach, not stating much more than that they have an *entity extraction module*, *disambiguation module*, *entity classification module*, and an *semantization module* (see Figure 3.7). Unfortunately, they do not supply much details of these modules. However, it is worth noting that the same authors appear in other papers related to the same topic, but it is not stated whether those more detailed explained methods are implemented in THD, or not.

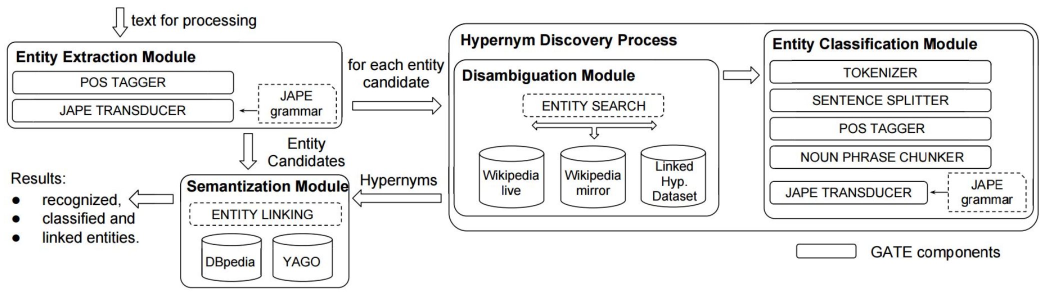


Figure 3.7: Architecture overview of THD.

They do, however, have a small comparison of their implementation against DBpedia and AIDA (see Section 3.2.3 and 3.2.4, respectively). THD supposedly perform real-time mining, i.e., once an entity is disambiguated to a Wikipedia article, the system extracts the hypernym from the article's free text. This allows the system to adapt to recent changes in Wikipedia.

Since THD extracts the types from free text, it is often complementary to the types of more semantically structured knowledge bases, such as DBpedia. THD returns both the mined type, and types from DBpedia and YAGO. The complementary character of the results can be utilized for classifier fusion.

Web Service

In order to use their REST API, one must obtain a free API key. This key is obtained by submitting a short request form on their Web site. A key will be granted as long as the user intend to use the service for evaluation, research and/or teaching purposes.

The API key is the only required parameter to be sent along with a request, but also this API supports several parameters for fine-tuning the whole process. This includes choosing language, knowledge base, linking method, spotting method, etc. The input text is sent as POST data.

When testing the API, it managed to correctly annotate *"Gerrard"* with an absolute confidence, *"Liverpool"* was wrongly annotated to the city with a confidence value of 11%, while *"Carragher"* was weirdly only annotated as a named entity, without any further information.

3.2.8 Other Candidates

There are lots of other entity linking systems that seem to be quite well-performing judging by their demos. However, many of these are commercial, allowing only a certain amount of free API calls, and when this threshold is exceeded the user is charged for a subscription.

Since the goal of a commercial actor is to earn money, they do not want to share too much detailed information about their technical approach with their competitors either. Therefore, obtaining a decent amount of well-defined documentation of their system is often challenging.

Rosette[20] is an entity linking system which by default uses Wikipedia as knowledge base, but also allow users to use their own custom database of any kind. They provide up to 10,000 free calls to their API each month, but charge money after this. This system was omitted from our selection due to its lack of documentation regarding their approach.

Microsoft Entity Linking Intelligence Service[21] is a part of Microsoft's *Cognitive Services*, concentrated around multiple tasks typically within the field of artificial intelligence. They use Wikipedia as knowledge base, and allows 1,000 API calls per day. How to use the API is well documented, with an easy *Getting Started Guide*, unfortunately there is not much information to obtain about their technical implementation of the system, forcing us to omit this system.

[20] `https://www.rosette.com/function/entity-linking/`

[21] `https://www.microsoft.com/cognitive-services/en-us/entity-linking-intelligence-service`

There are also other systems, which either has a slightly different purpose, or lacks a satisfyingly documentation, such as ***Open Calais***[22], ***Ontos***[23], and ***Alchemy API***[24] by IBM.

It is also worth noting that many entity linking systems make use of some of the systems already mentioned. For instance, ***Dexter***[25] is using an implementation of TagMe (see Section 3.2.2), where the only difference between Dexter and TagMe is the spot extraction methods and Wikipedia dumps being used. Earlier we have also mentioned ***FOX***, which uses AGDISTIS to disambiguate entities (see Section 3.2.5).

[22]`http://www.opencalais.com/`
[23]`http://ontos.com/`
[24]`http://www.alchemyapi.com/`
[25]`http://www.dxtr.it/`

Chapter 4

Approach

4.1 Theoretical Solution

Many of the entity linking systems we have previously studied, have supported multiple methods, techniques and algorithms to annotate the input text. Most systems also use their own custom algorithm for finding an entity in the text, called Named-Entity Recognition (NER). NER is a task that seeks to locate, and often classify, named entities in written text (see Section 2.1.1).

It is beyond the scope of this work to implement an entity extraction module from scratch. Instead, we use an existing NER tool to find entities for us. Our task is to disambiguate found entities, considering the context around their appearance. We detect how well our entity extraction module performs according to the real world, and possible deviations from reality, as we evaluate the system.

As earlier mentioned, building a platform that can fully understand unstructured written text is beyond the scope of this project. Thus, we first and foremost use the found entities in order to disambiguate each other. This means that our context will mainly depend on the entities found by the NER tool, and these will be checked for relations and similarities using a knowledge base consisting of structured information.

In order to find the best candidate for ambiguous cases, we introduce a scoring function telling how probable it is for each candidate to be the correct candidate in the given context. The context consists of four neighboring entities (or less, if there are not enough entities).

This neighborhood property is usually a symmetric relation, i.e., if entity A is the neighbor of entity B, entity B is also be the neighbor of entity A. However, this is not the case for the first and last couple of entities in the text, as they do not have the necessary amount of entities on both sides (see Figure 4.1). All entities will also be populated with a set of subjects describing them.

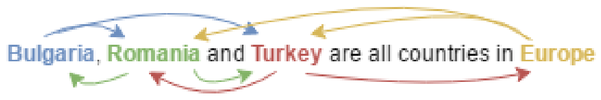


Figure 4.1: A text with four entities, here with just two neighbors. Romania and Turkey are neighbors of Bulgaria, and so on.

Our approach is based on the assumption that similar entities usually occur together in texts, and that similar entities have similar subjects in the knowledge base. Thus, if the mention *"Liverpool"* has footballers as neighbors, it is probably meant to represent the football club rather than the city. However, if it has other cities as neighbors instead, it is probably meant to represent the city. By checking the subjects of each entity candidate up against each other, we hope to be able to find a theme in the text, and use this to iteratively select the most fitting candidates.

We can split our system into three different modules: *entity extraction*, *candidate extraction* and *entity disambiguation* (see Figure 4.2). The entity extraction module detects and extracts all entity mentions in the text, before the candidate extraction module finds all potential candidates located in a knowledge base for each entity mention. Finally, the entity disambiguation module decides what candidate should be used for each entity.

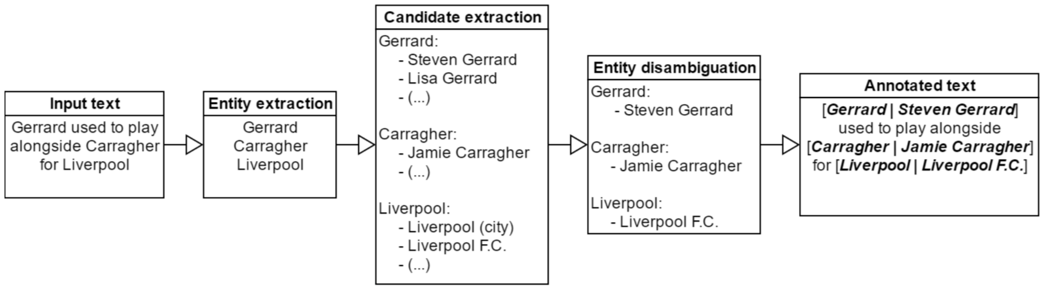


Figure 4.2: Example use of our proposed system.

4.2 Methodology

The work described in this report is heavily based on work conducted in the specialization project, autumn 2016. Through that work, we gathered information about entity linking, and fields of study somewhat related to entity linking. We especially took a deeper look at state-of-the-art approaches, and studied different well-known and widely used entity linking systems. Therefore, we had a clear perception of the problem before we started the work presented in this thesis.

We perform simple initial testing as we explore our hypotheses and develop the system, to ensure that our approach works. This step consists of annotating short custom designed text fragments, consisting of both more and less ambiguous entity mentions, but where the correct entity candidate should appear to be rather obvious. We think this is valuable in order to find out if our approach is applicable as early as possible, allowing us to modify our approach early in the development process if necessary.

Finally, we perform a more formal evaluation of our system. At this point, we identify, retrieve and parse a publicly available dataset designed for evaluating entity linking systems. As a result on these experiments, we calculate our system's precision, recall and F measures, and compare them to those of other systems.

4.3 Knowledge Base

We want to use a knowledge base (KB) covering a wide range of topics to be well prepared for covering the entities we discover. However, it might be even more important that the KB provides well-defined and structured data, which is easy to handle for a computer. Semantically structured data simplifies the process of finding relations between entities, since they are already encoded as part of their object representation in the KB.

We use DBpedia as KB, which is a widely used KB with structured information gathered from Wikipedia (see Section 2.3.2). Their system is well documented, and having a wide user base they naturally also have a big community available to help out if any problem should occur. DBpedia's close integration with Wikipedia, ease of use, and the fact that it is a very prominent KB in the Linked Open Data cloud, are all important factors as we choose DBpedia over other knowledge bases.

All entities in DBpedia contains a link to the Wikipedia page they originate from, so as we assign a DBpedia article, we indirectly also assign a Wikipedia article, since DBpedia articles and Wikipedia articles has a one-to-one relation.

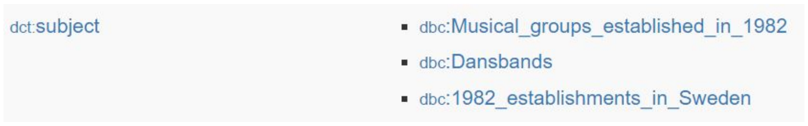
4.3.1 Categorization

The knowledge stored in DBpedia can be used to help disambiguate entities, ensuring that each entity points to the most relevant resource in DBpedia. The entities in DBpedia are described with attributes such as *"type"*, *"subject"*, *"birth place"*, *"population"*, etc. These attributes may be of help when we want to find relations between entities. We assume that similar entities are described with similar attributes, thus we can quantify how similar two entities are based only on their attributes.

What kind of attributes are present for each entity depends on what kind of object it is representing. We use the *"subject"* attribute in our disambiguation process, which should be present for all entities. The subject attribute consists of a list of categories used by Wikipedia to classify and give some general information about each entity. We believe these categories are well suited for disambiguating entities.

4.3.2 Erroneous Data

Figure 4.3 shows that knowledge bases such as DBpedia are not necessarily flawless. The English Wikipedia page classifies *Contrazt* as a Swedish band established in 1982, even though they are in fact a Norwegian band established in 2004, as they are correctly classified as on their Norwegian Wikipedia page. It is however worth noting that both the Norwegian and English version have roughly the same information in plain text (Norwegian band established 2004 for both), but since DBpedia use infoboxes and metadata (e.g., categories) from the English Wikipedia page, the information supplied by DBpedia are erroneous.



dct:subject	▪ dbc:Musical_groups_established_in_1982 ▪ dbc:Dansbands ▪ dbc:1982_establishments_in_Sweden

Figure 4.3: Subjects connected to the dance band *Contrazt* in DBpedia.

4.3.3 Access

DBpedia serves one live SPARQL endpoint[1] representing Wikipedia with a small delay of at most a few minutes, that is considered as the semantic web mirror of Wikipedia [31]. The main objective of DBpedia Live is to keep DBpedia always in synchronization with Wikipedia. Additionally, they serve one endpoint where the data sets are refreshed every once in a while (usually 1-2 times a year)[2]. As of February 20, 2017, the last data dump stems from April 2016, meaning we risk that many entities are outdated.

The difference between the live version and the standard version becomes apparent if we look at the page of *Donald Trump*, the current President of the United States. Using the standard endpoint (with data from April, 2016) he is classified as *"United States presidential candidates 2016"*, but naturally without any additional information that he in fact is the current president. Looking at the live endpoint however, he is also classified as *"Presidents of the United States"* and *"Republican Party Presidents of the United States"*.

[1]`http://live.dbpedia.org/sparql`
[2]`http://dbpedia.org/sparql`

In situations like these it is obviously an advantage to use the live endpoint, since we otherwise would lose vital new information.

In a commercial environment, one should of course use an KB as up-to-date as possible, but for initial testing and evaluation it is quite valuable to use a static KB you know will not be in constant change. Knowing that the KB does not change simplifies the testing step considerably, since we know that our implementation is the only thing that may vary for each program execution. Therefore, we will see what works and what does not regarding parameter values, and minor changes for formulas and functions, etc.

4.4 Description of Implementation

As earlier mentioned, our system can roughly speaking be split into three modules: *entity extraction*, *candidate extraction* and *entity disambiguation* (see Figure 4.4). Our contribution is mainly towards the entity disambiguation task, located within the entity disambiguation module.

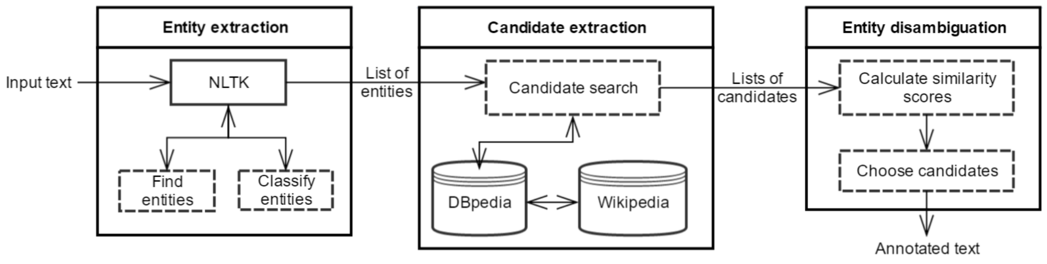


Figure 4.4: Architecture overview of our proposed system.

We want these modules to be as independent of each other as possible, thus a modification to one of them should not make any severe impact on the others. We also want the system to be both effective and efficient, but without any extreme focus on one over the other. For efficiency, we simply demand that the system does not spend too much time annotating the text, making it impractical for the user. Regarding effectiveness, we want the system to be the best possible, without breaking our demands for efficiency.

For more details about our implementation, take a look at Appendix A.

4.4.1 Entity Extraction

As mentioned earlier, we do not implement our own analyzer to extract the entities mentioned in the input text, but rather use an existing Named Entity Recognition (NER) tool. We use *NLTK 3.0*[3] [3], which contains an interface to Stanford NER [15] for Python. NLTK is available in the *Python Package Index*[4].

The input text we want to annotate is initially processed using NLTK. First, the text is tokenized and split into single words, before part-of-speech (POS) tagging is performed. The POS tagger processes a sequence of words, and attaches a part-of-speech tag to each word, telling whether the given word is a noun, verb, adjective, preposition, etc. Finally, we use a named entity chunker, which should classify all named entities in the text.

There are several potential approaches for doing this more specifically, but all should follow pretty much the same set of steps as mentioned. However, minor differences may lead to slightly different results. How we choose to classify the entities using NLTK is one example of such. We could use *binary classification*, which simply tells if the given word is a named entity or not, or we could use a *multiclass classifier*, which states if the entity describes a person, organization or geo-political entity. In theory, one might assume that both these approaches would result in the same amount of entities, however this is apparently not necessarily the case.

To demonstrate this, we use the test sentence *"Gerrard used to play alongside Carragher for Liverpool"* again. *"Gerrard"*, *"Carragher"* and *"Liverpool"* are all marked as *"person"* entities when using multiclass classification. *"Liverpool"* should not have been mapped to person, but is annotated as a named entity nonetheless. When using binary classification however, *"Carragher"* is not annotated at all, even though both *"Gerrard"* and *"Liverpool"* are correctly annotated as named entities. Because of this, we choose to use multiclass classification, but to ignore the actual classification and just treat them all like any named entity.

[3]`http://www.nltk.org/`

[4]`https://pypi.python.org/pypi/nltk`

4.4.2 Candidate Extraction

As all named entities in the text are discovered, a list of all entity mentions are passed over to the candidate extraction module, responsible to make and populate the objects, in order to make them more manageable. All entity mentions discovered by the entity extraction step will be represented by an *Entity* object.

The list of candidates for an entity will typically consist of all entities mentioned in the correct entity's disambiguation page we retrieve from DBpedia using their SPARQL endpoint. For instance, Gerrard's disambiguation page[5] consists of links to articles about Steven Gerrard, Lisa Gerrard, and Alfred Gerrard, to mention a few. Even though we do not know the correct entity at this point, it is in most cases relatively straightforward to find the correct disambiguation page given only the entity mention.

As we retrieve all these entity candidates, we also specify that we want all their corresponding subjects via the SPARQL query. With this information, we make *Candidate* objects for each candidate. Initially all candidates' probability score (S) are evenly distributed:

$$S_i = \frac{1}{number\ of\ candidates}$$

If we do not find a disambiguation page of an entity mention, we retrieve only the page matching the entity mention (or automatic redirects of it) if it exists. It is also worth repeating that our Candidate objects are representations of the actual entities in DBpedia, while our Entity objects are representations of the entity mentions in the input text.

The SPARQL request is built to return a list of candidates that could match the entity mention, rather than returning the most probable candidate at once. For instance, if the entity mention is *"Liverpool"*, we want to retrieve all candidates that could potentially represent it, rather than the default resource that represents Liverpool the city. It is fair to assume that the default resource is accurate more often than not, but it is certainly not always the case. Thus, we retrieve all potential candidates, and decide which one to use afterwards.

[5]`http://dbpedia.org/page/Gerrard`

4.4.3 Entity Disambiguation

Our main contribution is towards entity disambiguation, i.e., choosing the correct candidates considering the context around their appearance. This is achieved by comparing the candidates' subjects with each other, looking for similarities (see Figure 4.5). When we find entities that appears to be similar, we assume that they are probable to be the correct entities in that given context, and give the candidates a score accordingly.

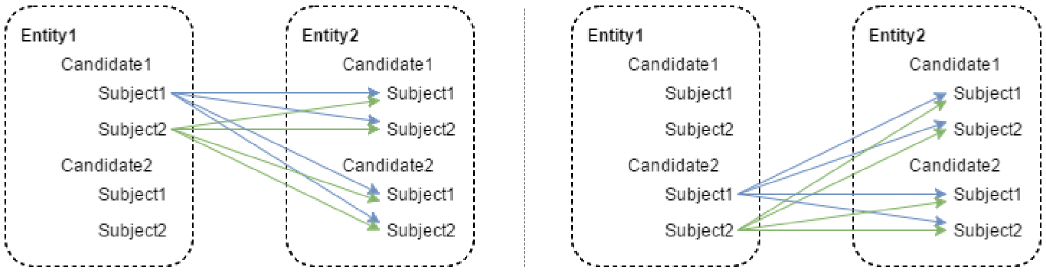


Figure 4.5: Overview of scoring process for *Entity1*, having *Entity2* as neighbor. All arrows indicate a string comparison that returns a score. All these scores are summed up to a total score for that given candidate.

Our approach is to iteratively recalculate the scores of each candidate by putting more weight on candidates we think have a higher probability of being the correct candidate for an entity mention, a process that we will take a closer look at in a moment.

As mentioned earlier, the candidate extraction module will evenly distribute the probability of all candidates belonging to an entity mention, meaning that all candidates initially have the same score (or weight) that sums up to one. This also means that all candidates will be equally important for the first iteration. For the remaining iterations, we scale the scoring according to the candidate's score.

If we are to recalculate the candidates' scores for an entity *E1*, whose neighboring entity *E2* have the candidates *C1* (score 0.8) and *C2* (score 0.2), the scoring process will put more emphasize on similarities between candidates of *E1* and *C1*, than *C2*. More specifically, the scoring function will multiply the score obtained with *C1* with 0.8, while the score obtained with *C2* will be multiplied with 0.2. This way, we ensure that we put more weight on candidates we believe are more probable of being the correct candidate for its entity.

When we recalculate a candidate's score, vi iterate through every candidate of each neighboring entity. At this point, we compare the subjects attached to the actual candidate whose score we are updating, and the subjects attached to the candidates of the neighboring entities (see Algorithm 1). We compare the subjects and give them a score according to the length of their longest common substring.

```
/* Variables with a leading n indicate that it is a
   neighbor of the entity we are currently working on */
foreach cand in candidates do
    cand.score ← 0.0
    foreach nEntity in neighbors do
        foreach nCand in nEntity.candidates do
            candPair ← 0.0
            foreach subj in cand.subjects do
                foreach nSubj in nCand.subjects do
                    candPair ← candPair +
                     longestCommonSubstring(subj, nSubj)
                end
            end
            cand.score ← cand.score + candPair · nCand.score
        end
    end
end
```

Algorithm 1: Pseudocode of scoring process.

Since we may have to compute many cases of the longest common substring (LCS) problem, this is a critical part of our implementation regarding efficiency. Our algorithm is a somewhat simplified version of the *longest common subsequence* algorithm using dynamic programming presented by *Cormen et al.* [7]. A brute-force approach to the LCS problem would take exponential time, which is not satisfying in our case. The running time of our algorithm however, is $\Theta(mn)$, where m and n are the lengths of the input strings we want to compare.

We are able to simplify the algorithm since we only need to find the strictly common substring, i.e., the input strings *"ABC"* and *"ABXC"* should result in length 2 (*"AB"*), not 3 (*"ABC"*). Additionally, we only care about the length, and do not need to reproduce the actual sequence as suggested by Cormen et al.

Candidates that have neighboring candidates with similar subjects will get a higher score. Our implementation finds the length of the longest common substring of two actual subject-pairs across neighboring candidates, and divides this value with the product of the length of the two input subjects. We do this to avoid giving higher scores for longer subjects than we do for shorter, since longer texts naturally have a higher probability of having longer string matches. Additionally, we set the score to zero if the longest common substring has a length of less than five, to avoid scoring random string matches that sometimes occur in texts.

$$subject\ pair = \frac{length\ of\ longest\ common\ substring}{length\ of\ subject\ A \cdot length\ of\ subject\ B}$$

Since all candidates for an entity initially have equal scores, the candidates with the most "common" theme across all entity candidates should have the best score after the first iteration. For each iteration, we pick a winning candidate for exactly one entity. We choose the leading candidate that is most superior to the second best candidate across all entities with more than one candidate (see Figure 4.6). This process is repeated until all entities have just one remaining candidate.

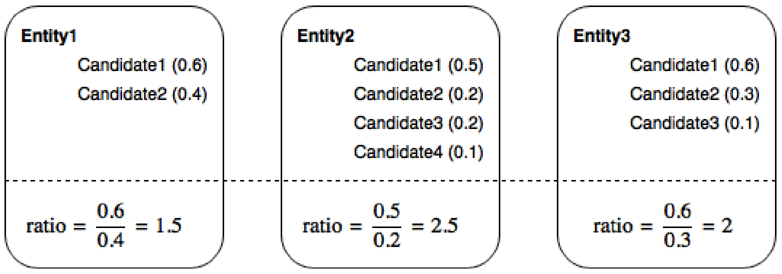


Figure 4.6: Three entities, each with 2-4 scored candidates. *Candidate1* will be chosen for *Entity2* this iteration since it has the highest winning ratio (2.5), even though both *Entity1* and *Entity3* have candidates with an higher individual score.

For each iteration, we discard all candidates whose score is worse than $\frac{1}{2}$

of the best candidate. This technique is beneficial regarding the system's efficiency, since we recalculate every remaining candidate for each iteration. When discarding candidates like this, we reduce the work needed to be done for future iterations.

By setting the discarding threshold to a higher factor, the system will execute quicker, but risk to discard the correct candidate due to a bad score in an early iteration, as the context is usually be most ambiguous initially. If we set a too low threshold value however, the program will not execute considerably faster.

When all entities have just one remaining candidate, we check if this candidate is the same as DBpedia's default resource for that entity mention (e.g., *"Liverpool (city)"* is default for the mention *"Liverpool"*). If one such default resource exist, but does not match our only remaining candidate, we add it to our candidate list and give both candidates a score of 0.5. All entities should now have one or two candidates, and we start the iteration process once again to find the correct candidates.

If the default resource is added again, the remaining iterations use the average score, i.e., the score is calculated as before, but now it is divided by the candidate's number of subjects. This is more fair since some candidates have very few subjects, while others have very many. It is natural that a candidate with 20 subjects get a higher score than a candidate with 2 subjects due to random matches. For the earlier iterations we ignore this, since candidates with more subjects are usually more popular, and are therefore generally more statistically probable to be the correct candidates. When we add the default candidate again however, both candidates are considered as relatively probable, so we want to select the candidate that has the best score per subject.

The reason for giving the default resource two chances, is that it is often, but not always, the correct resource. Thus, if the default resource has been discarded in an early iteration, we want to give it a new chance at the end, when we have a better understanding of the context. Since all entities at this point have either one or two candidates, this is not very resource demanding either. When all entities are down to having just one candidate again, the disambiguation process is complete and the remaining candidates are chosen for entity linking.

This whole disambiguation process could be very resource intensive, since a candidate often have over 20 subjects. The number of candidates for an entity may also exceed 20. Having multiple entities as well, this could easily become infeasible to perform in a serialized manner without exceeding time constraints and demands for an user of the system. Therefore, we run multiple parts of the disambiguation step in parallel, in order to speed up the process.

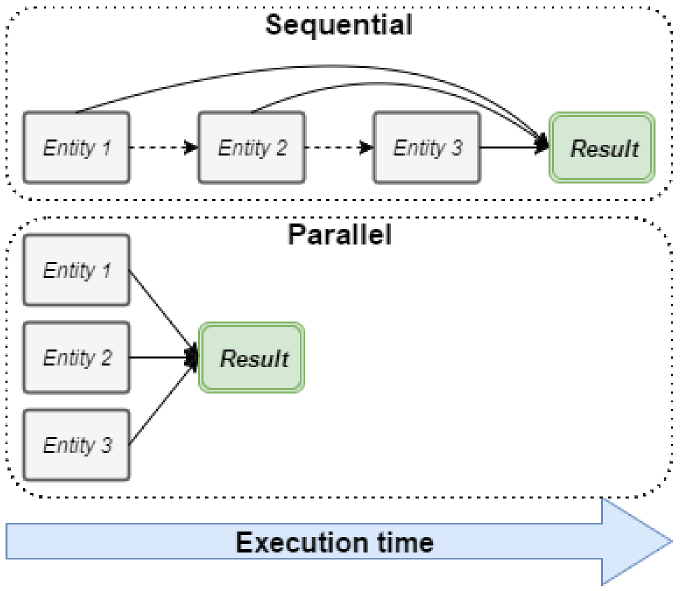


Figure 4.7: Sequential versus parallel approach for the disambiguation process.

First off, we disambiguate each entity in parallel, i.e., we initiate the disambiguation process for all entities at once, and the processor thereby disambiguate as many entities as possible simultaneously (see Figure 4.7). Additionally, we calculate the score for each candidate within an entity in parallel. These approaches enhances the execution time tremendously.

4.5 Initial Testing

As we implement the system, we also set up a couple of short texts used for testing the system's performance as we made minor modifications to our implementation. This way we could easily get an indication for how beneficial the new modifications were for the system's performance, regarding both effectiveness and efficiency.

These texts were not necessarily meant to be representative for ”normal” real-world text, but were written in a manner which makes them especially interesting in the case of entity linking. I.e., we wanted ambiguous entity mentions to see if we could annotate them correctly. Even though we wanted ambiguous entities, we also wanted the context to make it pretty clear which candidates should be selected, ensuring that the system was capable to correctly annotate ambiguous, but still rather obvious entities.

For these ambiguous cases, we also wanted to ensure that the system was able to select different candidates depending on the context. E.g., we used the sentences *”Gerrard used to play alongside Carragher for Liverpool”* and *”Gerrard is a musician from Melbourne, Australia”*, to see whether the system was able to select Steven Gerrard as candidate for the first sentence, and Lisa Gerrard for the second. Another example is the entity mention *”Pluto”*, which we used to see if the system was able to select between the dwarf planet or the Disney character considering the context.

We also tested with texts that should be rather straight forward to annotate correctly, such as *”Bulgaria, Romania and Turkey are all countries in Europe”*, etc. The default resource is usually the correct candidate for the entity mentions in these texts that are supposed to be simple. By using both ”simple” and ”complex” test sentences, we assured that the system could annotate simple, as well as more challenging entities correctly.

We managed to iteratively improve our system's capability to annotate the input texts, and discovered that small variations with the implementation could be beneficial for some texts, while being detrimental for other texts. Thus, we found that it would be rather optimistic to think we could find an approach that would work ideally for all cases, and we figured we would be better off concentrating on finding an satisfying approach for all (or as many as possible) cases, optimizing the overall performance.

However, it is worth noting yet again that these tests were not meant to measure the system's performance for real-world applications, as the text in such cases will be more rich than what was the case for our minimalistic test sentences. We also discovered that NLTK had some problems discovering all entities in some texts, so we had to formulate the texts to assure that all entities would be found, as this was essential in order to measure the effectiveness. This would obviously not happen in a real world application.

The efficiency was also measured, as we instantiated a timer to see how much time was needed to annotate our texts. This proved that implementing concurrency when calculating scores had a very positive impact on the program's execution time, without any impact on the effectiveness, of course. For a test with four entities (*"Gerrard"*, *"Melbourne"*, *"Australia"*, and *"Oceania"*), the annotation process was around 20 times faster with an simple parallel approach than when doing everything in a serialized manner. It also showed what was the better method to find the longest common substring, among other details regarding our implementation.

Chapter 5

Evaluation

5.1 Experiments

In order to measure our system's capability to annotate text, we want to evaluate its performance by automatically annotating text that is supposed to be relatively close to a real-world domain. In order to quantify the results, we need a fairly big amount of text to cover many scenarios, as well as a solution for how this text ideally should be annotated. With this in place, we can check how many annotations our system succeeds and fails to produce.

5.1.1 Dataset

We evaluate our system using the Wiki-Annot30 dataset[1], developed by the same team who developed the TagMe entity linking system (see Section 3.2.2), and used to evaluate their system. As the dataset is publicly available, other systems have used it for evaluation as well.

The dataset contains short text fragments drawn from Wikipedia snapshots of November 6, 2009. The text fragments are composed by about 30 words, and they contain about 20 non-stopwords on average. Each fragment contains at least one ambiguous entity mention. Even though the data are somewhat aging, it is still as relevant as before in the case of entity linking.

[1]`http://acube.di.unipi.it/tagme-dataset/`

The Wiki-Annot30 dataset consists of 186K text fragments that is formatted with the following syntax: the first line contains the actual text, the second line contains a list of gold standard entities, i.e., all entity mentions present in the text, followed by numeric IDs of the articles which they are pointing to (see Figure 5.1).

```
Czechoslovakia, 1982), The Trap Door (UK, 1984). Films include,
Chicken Run and The Adventures of Mark Twain. Cutout animation
is a type of stop-motion animation produced by moving 2-dimensional
czechoslovakia 5322 films 21555729 stop-motion animation 27036
the adventures of mark twain 11484373 uk 31717 2-dimensional 35248
stop-motion 27036 the trap door 1604203 cutout animation 745626
chicken run 284525
```

Figure 5.1: Text fragment with corresponding annotations retrieved from the Wiki-Annot30 dataset.

5.1.2 Hardware

Annotating a rather large amount of the Wiki-Annot30 dataset with our system would be infeasible on our aging Windows 10 laptop with limited resources (one processor with two 1.70 GHz cores, each with four threads). This gave us the option between annotating a very small part of the dataset, or to obtain a more powerful computer to perform the evaluation process on a bigger part of the dataset. The latter would obviously be the better solution, so we requested access to a more powerful machine by NTNU IDI Drift, which was granted.

We connect to this computer using PuTTy[2] and Secure Shell (SSH). This computer runs on Ubuntu 16.04, and consists of twelve processors, each with six cores with a clock rate of 3.50 GHz, and each core has two threads. This computer is obviously superior to our laptop when it comes to computational power. Even though this machine may have several users simultaneously, it is fair to assume it will perform considerably better than our laptop.

[2]`http://www.putty.org/`

In order to get an idea of how different the two machines perform, we conduct a very simple experiment where we iterate through a loop 100M times. For each iteration, we increment a numerical variable defined to be zero before the loop. We measure the time needed for both our laptop and the server to complete the task. Our laptop needs 342 seconds (almost 6 minutes), while the server only needs 6 seconds. This is a very simple serialized test, and does not take into consideration how the two machines would perform with parallellized approaches, but it is no secret that our laptop is inferior in most situations, if not all.

5.1.3 Evaluation Metrics

We measure our system's effectiveness by calculating its precision, recall and F measures. The precision measure tells how many of the annotated entities are in fact correctly annotated. Recall is used to measure our system's performance against the gold standard, i.e., how many entities in the gold standard are also correctly annotated. Finally, we find the F measure, that represents the harmonic mean of precision and recall.

$$Precision = \frac{\#\ of\ correctly\ annotated\ entities}{\#\ of\ all\ annotated\ entities}$$

$$Recall = \frac{\#\ of\ correctly\ annotated\ entities}{\#\ of\ gold\ standard\ entities}$$

$$F = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

We believe that our system will produce better results for precision than for recall, considering we have put more effort into entity disambiguation than entity extraction. Additionally, the gold standard consists of entities beyond just named entities, which we focus on, and thus consists of considerably more entities than what we expect to extract. With this in mind, our system would probably be better off using the $F_{0.5}$ measure, which puts more weight on precision than recall. However, we stick to the F_1 measure, since that is most used, and is generally more fair. This also simplifies the process of comparing our system against others.

5.1.4 Evaluation Methodology

First, the dataset consisting of all the texts we want to annotate must be parsed. Each text fragment is put into an *AnnotatedText* object, alongside a dictionary consisting of all entities in the text, and their Wikipedia IDs, like this: *"{gerrard: 547384, carragher: 1012020}"*. Thereby, we have an understanding of how a perfect system should annotate the text according to the gold standard.

We now begin to annotate the text we have parsed from the dataset. This is simply achieved by looping through all AnnotatedText objects, and performing our entity extraction and disambiguation algorithm on each object's text property. When the disambiguation process is complete, and our system have annotated all found entities in the input text, we compare our findings with the suggestions encoded in the dataset.

Since the entities in the dataset are manually annotated (as it is a Wikipedia snapshot), while we use NLTK for entity extraction, our entity set is often different from that of the dataset, as we would expect. Many entities are simply not found, and some are slightly different. E.g., the dataset has an entity named *"The Trap Door"*, while our system finds the entity *"Trap Door"* instead. Sometimes, what is actually one entity is split into multiple entities, e.g., *"United States of America"* might be split into *"United States"* and *"America"*. These kinds of errors may not always have a significant meaning, but we risk losing important information when errors such as these occur.

Because of this, it would be unfair to demand that the string representations for the entities we find matches the string representations in the dataset. Instead, we check how many of our found Wikipedia article IDs are also encoded in the gold standard for the same text fragment. If the same ID is present both places, our system gets one correct annotation, otherwise it does not.

Additionally, we keep track of how many entities are present in the gold standard, how many entities our system finds, how many of the entities we found are exact string matches from the gold standard, and how many candidates we retrieve overall.

These values are used to calculate our system's score for precision, recall and

F measures (see Section 5.1.3). We use these measures to obtain an overview over the system's effectiveness. This also allow us to compare our solution to other existing systems that has been tested on the same dataset, using the same evaluation metrics.

We perform the experiments by connecting to the machine we got access to through NTNU Drift, which is considerable more computationally powerful than our laptop, using SSH. We set up our program on this computer in order to evaluate a large (but still practically feasible) part of the Wiki-Annot30 dataset we have selected for evaluation in a feasible manner.

5.2 Results

We evaluate our system by running the first 1,000 text fragments from the Wiki-Annot30 dataset through our algorithm and see how many of our annotations matches the suggested annotations in the dataset, also referred to as the gold standard. Even though we use a powerful computer, we still need over an hour to automatically annotate all the texts with our system.

5.2.1 Entity Extraction

The gold standard suggests that there are 5,874 entities in the first 1,000 texts. Our system ends up identifying 3,762 entities, where 1,500 are exact string matches to the gold standard (see Table 5.1). These 1,500 entities that are completely correctly extracted, makes out 25.5% of the gold standard entity set, and 39.9% of our extracted entity set.

Entities in gold standard	**Extracted entities**	**Exact matches**
5,874	3,762	1,500

Table 5.1: Results of our entity extraction module over 1,000 text fragments.

Altogether, we extracted 65,626 candidates for our 3,762 entities, meaning the entities in average have about 17 candidates each initially.

5.2.2 Entity Disambiguation

Our system manages to annotate 1,694 entities similar to the annotations in the gold standard, meaning that 2,068 annotations are considered to be erroneous (see Table 5.2). This also means that we are able to correctly annotate several entities that are solely not string matches of those listed in the gold standard as well, since only 1,500 of our entity mentions are identical to those of the gold standard.

Entities in gold standard	**Extr. entities**	**Correct annotations**
5,874	3,762	1,694

Table 5.2: Results of our entity disambiguation module over 1,000 text fragments.

With these numbers, we obtain a precision score of 45.0%, recall score of 28.8%, and F score of 35.1%, using the formulas presented in Section 5.1.3. It is worth noting that our system extracts and correctly annotate an undefined number of entities that are not suggested in the gold standard as well. However, these occurrences are only found by manual inspection, and are therefore not counted towards correctly annotations in this evaluation.

Chapter 6

Discussion

6.1 Dataset

The Wiki-Annot30 dataset consists of 186K text fragments drawn from Wikipedia, each composed by about 30 words. Each text fragment consists of approximately six entities, while our experiments show that we on average manage to extract almost four entities per text fragment. Since our system supports four neighbors per entity, we would prefer longer texts, with more entities. This would give us a better view of the context in each case.

Some text fragments consist of very few entities (e.g., 2), therefore we risk ending up with none, or very few entities in some cases. If we do not find any entities there is nothing for us to do, if we find one entity there is no context to consider, as we need to find at least two entities in order to use our disambiguation process. We could choose to ignore text fragments where we find very few entities, but we do not, as we want to see how our system perform in "all" kinds of situations.

One might as well argue for selecting a dataset that have used the same entity extraction techniques as we use, ensuring that we would find the exact same set of entities. An alternative could be to use a dataset with pre-defined entities, where all entities would be explicitly marked as entities beforehand. Using the latter method, we would solely focus on the disambiguation process when evaluating, as the entities we want to extract would already be encoded in the dataset.

However, we found it more desirable to test our system in a more realistic domain, which is closer to a real-world application. We think the Wiki-Annot30 dataset is well suited for such experiments, and therefore a good selection, even though somewhat longer texts would be preferable.

6.1.1 Aging Content

The dataset is a snapshot from Wikipedia as of November 6, 2009. Being over eight years old, Wikipedia has naturally evolved with a lot of new data over the years, meaning that the Wikipedia version used for the dataset and the one we use are quite different. During this eight year time span, some Wikipedia articles may have been deleted, moved, added to other articles, gotten a new ID, etc. Changes such as these might make it harder, or even impossible, for us to produce the same link again today in some cases.

6.1.2 System Comparison

The Wiki-Annot30 dataset has been used for evaluating other entity linking systems as well, giving us the possibility to compare the effectiveness of our system with other systems. We compare our scores with the scores of *Wikify!* (see Section 3.1.1) and *TagMe* (see Section 3.2.2), their scores are illustrated in Table 6.1.

	Precision	**Recall**	**F-Measure**
TagMe	76.3	76.1	76.2
Wikify!	69.3	69.5	69.4
Our System	45.0	28.8	35.1

Table 6.1: Our system compared to other state-of-the-art entity linking systems.

Our approach is not especially designed to be used with this dataset, or vice versa. Much of our problems are due to our method for entity extraction, as we often end up with a different entity set than what is the case for the gold standard. Our disambiguation process heavily depends on which entities are extracted, thus any deviations between our entity set and the one of the gold standard will have a severe impact on our capability to achieve correct annotations.

Thus, if we are able to improve the entity extraction step, it would also have a positive impact on the disambiguation process, resulting in more correct annotations, and better scores for precision, recall and F-measure.

6.2 Knowledge Base

There are several knowledge bases open for free use today, some of whom we have studied earlier (see Section 2.3). We selected DBpedia, due to its large amount of structured data, freshness, and easy accessibility through SPARQL. Even though it was of great use as we disambiguate the entities, we did experience some problems regarding inconsistency, etc.

6.2.1 Inconsistency

The lack of up-to-date data is a problem that quickly arises as we use the static endpoint. E.g., at the time of writing, Donald Trump, the current president of the United States of America, does not have any data stating that he in fact is the president of the U.S. This only goes for the static endpoint, as the live endpoint is updated with more accurate information.

It is not therefore said that the live endpoint is always correct, as erroneous data can be found several places for different reasons. Some data are simply wrongly entered from the start, e.g., *Contrazt* is stated to be a Swedish band established in 1982, while it actually is a Norwegian band established in 2004. Errors such as these will be present until someone update their classifications in Wikipedia, and DBpedia update their static dataset.

There are also some inconsistency in how the entities are described in DB-pedia. We expect similar entities to be described in similar manners, but this is not necessarily always the case. *Norway* has a property *"Countries in Europe"* among its subjects, and we expect that all other countries mention their continent among their subjects as well. However, *Australia* does not have any subject stating what continent it belongs to.

Inconsistencies such as these makes the disambiguation process more complex, since not all entities are described in a similar manner, even though we want to evaluate them in a similar way. What kind of subjects are connected to each entity decides the outcome of our disambiguation process, so we would strongly prefer a knowledge base with a pre-defined standard with strict rules for what kind of subjects that should be used for each situation.

6.2.2 Disambiguation Pages

Our implementation is based on initially ignoring the ”obvious” solution, but rather fetch all possible solutions. This is achieved by looking for the entity mentions' disambiguation pages, rather than pages that is exact string matches. In many cases, this lead to an impractical large set of candidates. Many of these are in reality rather obscure and unlikely to occur, but are nonetheless evaluated in the same manner as more ”normal” and regularly used candidates.

As we lack any measure of the candidates' general popularity, our system sometimes tend to select obscure candidates, which is so obscure that they could just as well have been left out from the start. As mentioned, however, we do not have any measure for general popularity to each candidate, meaning we have to include all, including the obscure and in reality unlikely candidates.

DBpedia's disambiguation pages are also somewhat inconsistent in some situations. Using Donald Trump as an example again, his DBpedia article lists *”Donald”* and *”Donald_Trump_(disambiguation)”* as his disambiguation pages. We expected that also *”Trump”* would be a disambiguation page, but this is apparently not the case. However, *”Trump_(disambiguation)”* is listed as a disambiguation of *”Trump”*, and *”Donald_Trump_(disambiguation)”* is listed as a disambiguation of *”Trump_(disambiguation)”*, meaning there is a path from *”Trump”* to the actual page about Donald Trump, even though it is not very obvious.

These sometimes inconsistent and complex structures of disambiguation pages aggravates the problem of finding all candidates for an entity mention, as we need to prepare for alternative paths as well, rather than consequently using the standard name followed by *”_(disambiguation)”*.

6.2.3 Access

For each entity we find in the text, we need to send a request to DBpedia's SPARQL endpoint. When we have very many entities, we also have to send a large amount of SPARQL requests, which might be a problem as we also have to wait for a response to all of them.

Experiments show that sending and receiving a response takes approximately 0.1 seconds, with some small deviation. The live SPARQL endpoint is somewhat slower than the static endpoint, but both use about 0.1 seconds on average. This means that during our experiments, where we found 3,762 entities, we grossly calculated used 6 minutes just by sending and receiving SPARQL requests.

Even though 6 minutes is not very much, considering the whole entity linking process takes over one hour, it would be beneficial to shrink it even more. This could be done by setting up a local DBpedia mirror using Virtuoso, a scalable cross-platform server that combines Relational, Graph, and Document Data Management with Web Application Server and Web Services Platform functionality[1].

If we set up our own DBpedia mirror locally, we can retrieve data from this local SPARQL endpoint, which is obviously time saving versus the option to request data from the official SPARQL endpoint. Considering we would need to serve the whole DBpedia dataset, we would need a very powerful machine in order to get a decent enough performance.

Even though we believe the external computer we used for evaluation is powerful enough for hosting our own local DBpedia mirror, we have not implemented this, as the current solution with using the official endpoint is not a big problem. Even though we could save some time, it is certainly not the current bottleneck regarding efficiency.

6.3 Entity Extraction

Since we did not implement our own entity extraction algorithm, much of the outcome from the entity extraction module was out of our hands. We

[1]https://github.com/openlink/virtuoso-opensource

chose to use *NLTK* (Natural Language Toolkit), a very popular platform for building Python programs to work with human language data. NLTK has the full responsibility of identifying and extracting entities from the input text. Even though NLTK was easy in use, and did pretty much what we wanted, it also had some flaws.

We experienced that the entity extraction step is a big problem regarding our system's effectiveness, as it is a big challenge not only to find entities, but also to group multiple words that should represent one entity, which we will get back to soon. In many cases, picking capitalized words is a good indication when looking for entities, but real world text is (unfortunately) way more complex than it would have to be in order for using only this technique to find entities with a satisfying outcome.

6.3.1 Multi-Word Entities

One reason for our entity extractions not matching the ones in the gold standard, might be due to our sometimes lacking capability of grouping mentions that consist of multiple words together. E.g., the dataset mentions an entity *"The Adventures of Mark Twain"*, which is a movie from 1985[2]. However, our system ends up with two entities: *"The Adventures"* and *"Mark Twain"*. Even though *"Mark Twain"* is linked to an article describing the person Mark Twain, which is sensible enough, our system do not get any correct annotations in this case, since *"The Adventures of Mark Twain"* is not annotated as expected by the dataset. We suspect that errors such as these appear quite often, and therefore have a severe impact on the system's effectiveness. In some cases we might also group more words than we should.

6.3.2 Entity Set

The annotations in the Wiki-Annot30 dataset are manually created by the users who contribute towards Wikipedia. Hence, it is humans deciding what are entity mentions and not in the text. In our case, we need to reproduce these results in a generative way programmatically, which is obviously a

[2]`http://www.imdb.com/title/tt0088678/`

challenge due to our lack of subjective judgment. This often leads to different entity mention sets for our system versus the gold standard.

While we focus on named entities in our entity extraction algorithm, many of the annotations in the dataset are more ”loose” concepts, like *”motion capture”*, *”stop-motion”*, *”cutout animation”*, etc. We do not look for entities like these, thus we naturally often end up with fewer entity mentions than what is the case for the gold standard. This also shows when we look at our recall score, which is 28.8%. We find 3,762 entities, while there is supposed to be 5,874. Of these 3,762 entities we extract, 1,500 (approximately 40%) are exact string matches of the suggested entities in the gold standard.

The other roughly 60% of entities we find, may not match the mentions in the gold standard for a number of ways. We end up annotating some entities which the dataset do not, e.g., our system correctly annotate the mention *”Philippine Revolution”*, which is not annotated in the gold standard. Cases such as these are only found by manual inspection of the results, thus our system do not get a point for a correct annotation in these situations. However, the entity mention is still valid, so these occurrences will have a negative influence when we evaluate our system. Even though errors such as these occur, we do not believe they appear very frequently.

6.3.3 Classification

NLTK allows us to extract entities with binary classification, just telling whether we have a named entity or not, or multiclass classification telling whether the entity is describing a person, organization or geo-political entity. We did however experience some problems when using only binary classification, as some entities weirdly were not annotated as named entities, even though they were annotated with multiclass classification. Thus, we use multiclass classification, but treat all classifications as any named entity.

6.4 Entity Disambiguation

Our main contribution is towards entity disambiguation, so we naturally explored several approaches especially for this. We experienced problems finding an approach that generally outperformed the rest, as their performance too a large degree were depending on the input text. If we managed to tweak our implementation to better annotate some texts, it often had detrimental effects on other texts.

6.4.1 Default Resource

As mentioned earlier, our approach is to evaluate all possible candidates on the same level, ignoring aspects such as popularity, title similarity, etc. Alternatively, we could pick the default resource where one such is present. In that case, the entity mention *"Liverpool"* would be linked to the article about Liverpool city, while *"Gerrard"* would still consist of a list of all candidates listed by its disambiguation page, since it does not have any default resource.

The good thing about this alternative approach, is that the default DBpedia is very often the correct one. Therefore, we avoid using much time on candidates that in the first place are rather unlikely to be correct. However, with this approach we certainly know that we will not be correct every time, e.g., the mention *"Liverpool"* will never be linked to anything else than an article describing the city, no matter what the context may be. This goes for all entity mentions that have a default resource in DBpedia.

When re-running the experiments we did earlier, but using this alternative approach with selecting the default DBpedia resources where they are present, our system's performance clearly improved. We were now able to annotate 1,892 entities correctly, over the original 1,694 we had. Our candidates set now consist of 42,702, considerably less than the original 65,626. Since we now have less candidates to handle, the execution time was also halved, to approximately 40 minutes. This gives precision, recall, and F scores of 50.3%, 32.2%, and 39.3%, respectively.

Even though this approach seems to be superior regarding both efficiency and effectiveness, the fact that we simply know it will fail a number of times is a big drawback. Thus, we prefer to stick to our original approach, since we believe that this approach have a better potential, even though there are some problems with the current implementation.

6.4.2 Initial Context

Our problem is greatest initially, as we usually have very little information about the context from the start. Some entities may have only one candidate from the start, meaning we can use it to get a better understanding of the context, but this is not necessarily the case in all situations.

The lack of knowledge about the context initially, makes it hard to select a reasonable candidate. As soon as we select a wrong candidate, the lack of knowledge about the context will propagate as the system will put more emphasize on the unknowingly wrongly selected candidate. When the harm is done, there is not really any way back, as the selected candidates will form our understanding of the context, and will play a big role when deciding which candidates to select next.

The problem of lacking a decent understanding of the initial context is unavoidable. However, we did try to minimize the inconvenience by not selecting a candidate for the first iteration, but rather just give all candidates a score. For the second iteration, we could then put more emphasize on the candidates considered to be most probable in the first iteration, and hopefully be more competent to make a good decision when actually deciding for a candidate.

Experiments showed that this approach with avoiding to select candidate for the first iteration did not have much influence on the outcome. Naturally, the system's execution time was extended, since we wait longer to discard candidates. Because of this, and that it did not improve our effectiveness notably, discarding this alternative approach was an easy decision.

6.4.3 Candidate Scoring

How the candidate scoring algorithm is implemented is a key aspect of our solution, as this in reality is the part deciding which candidates should be selected for linking. There are numerous ways to implement this in detail, where our approach is described in detail earlier (see Section 4.4.3). There are however some aspects that deserves some extra attention, which we will now take a closer look at.

Candidate Similarity

The key aspect of our disambiguation process is how we score similarity between candidates, giving us an almost unlimited amount of options. We went with a solution where we quantify the similarities between two candidates as how similar their subjects are according to a string matching algorithm. There are a number of aspects to consider as we implement one such algorithm, e.g., length of subject strings, and linear vs exponential scoring, which we will get back to in a moment.

First of all, the length of the subject strings vary, which is something we must consider when we score each instance, since longer strings obviously have a higher probability of getting longer common string matches and thereby a better score than shorter strings. In order to avoid this disproportionality, we divide the score given by our longest common string algorithm with the product of the lengths of the two subject strings. We thereby keep the ratio similar no matter how long the two subject strings are, and differently sized subject strings can be compared fairly. As expected, experiments showed that omitting this setting had deteriorating effects on the system's effectiveness.

Regarding how we find the longest common substring, i.e., the value we want to divide by the product of the lengths of the subject strings, we also have several potential approaches. We went with a simple linear approach, where the score simply reflects the actual length of the longest common substring of the two subject string. E.g., *"ABCDE"* and *"ABXC"* gives the value 2, since the longest common string (*"AB"*) is of length 2. In that given example, the following score is computed:

$$Score = \frac{2}{5 \cdot 4} = 0.1$$

An alternative to this linear approach of scoring longest common substring, is to use an exponential function. By using an exponential function, longer string matches will be much more important than shorter. The logic for using this approach is that longer string matches will typically indicate a more precise similarity than shorter common strings, and should thereby get a considerably higher score, in contrast to the linear approach where there is a smaller difference between the two. In detail, the alternative scoring function would look like this:

$$Exp.\ Score = \frac{base^{(length\ of\ LCS)}}{length\ of\ subject\ A \cdot length\ of\ subject\ B}$$

An important element to consider when using exponential scoring is what the base value should be. If it is too low, most cases will result in basically the same score. If it is set too high, longer string matches will quickly be very dominant, completely neglecting the impact from shorter string matches. Finding this sweet spot for the base value is a challenge. However, experiments showed that a base value of 1.15 worked well, and outperformed our linear approach marginally, as we managed to annotate 1,710 entities correctly, versus the original 1,694 with the linear approach.

We thought the ideal base value would be slightly higher than 1.15, which would cause a bigger increase of score as the length of the longest common substring increases. How the different functions influence the score is illustrated in Table 6.2. We notice that for the exponential approach (with base 1.15), the score stably increases with a factor of about 1.3 when the length of the LCS increases by two. For the linear approach however, the increase factor will converge towards one, i.e., the factor will decrease as the length of the LCS increases.

	Length of LCS				
	5	**7**	**9**	**11**	**13**
Linear	5	7	9	11	13
Exponential (base 1.15)	2.01	2.66	3.52	4.65	6.15

Table 6.2: Differences between linear and exponential scoring given the length of the longest common substring (LCS).

Common Words

No matter if we choose to use linear or exponential scoring, the candidates' subjects are pivotal for the outcome. E.g., if one candidate has a word occurring many times among its subjects, and some of the candidate's neighboring candidates have the same word occurring many times as well, these candidates will get a high score. This is really what we aim for, but sometimes this might be misleading. This could for instance happen if two neighboring candidates have the word *"american"* mentioned many times among their subjects. These candidates are then very probable to be picked by our disambiguation algorithm, even though this word might be the only word they have in common.

In such situations, one single word will have an unnatural big influence on which candidates are chosen. There are indeed a big chance that both candidates have some relation to America, but it would be preferable to find other similarities as well. A possible solution to this possible problem could be to use an reduction factor for words, or string sequences, that have already occurred, certainly if they have occurred a number of times. However, this technique is yet to be tested in practice, even though we believe it would have an positive impact on our solution.

Average Scoring

Our implementation initially calculates the score for each candidate without taking their amount of subjects into account. This means that a candidate's chance of getting a high score improves the more subjects it has. Our thinking behind allowing this, is that more popular candidates usually have more attached subjects as well, meaning our approach in practice actually will prioritize more "popular" candidates in many situations, in contrast to what we have earlier stated.

The assumption that often used candidates have more subjects than less used candidates are not necessarily always true, which may cause confusion sometimes. It is also worth noting that if we reintroduce DBpedia's default candidate for an entity in the end, we calculate the candidates' average score, i.e., their score per subject. This is because we then consider both candidates to be probable, and therefore do not want to give any of the candidates any kind of handicap.

Experiments show that this approach outperforms approaches based on always or never calculating the average score, which is as expected. Something more surprising however, is that never calculating the average score outperforms the option of always taking the number of subjects into account. This is not quite what we expected, but is again an indication that the most ”popular” candidate is usually the correct one.

6.4.4 Efficiency

Our system is not really as fast as we would wish, as the whole entity linking process on average needs around one second per entity on the quite powerful machine we used. How much time is needed obviously heavily depends on how many entities, candidates and subjects are present in the input text, as these are the factors deciding our program's execution time.

Longest Common Substring

The longest common substring, or longest common subsequence (LCS), is a problem we encounter very many times during our disambiguation process. This also means that it has massive influence regarding our efficiency, which also became clear as we experimented with different implementations for solving the LCS problem. Our current solution is a slightly simplified version of the state-of-the-art solution we have described earlier (see Section 4.4.3), which indicates that there are not an easy way to speed up solving this task considerably.

Parallelization

We have introduced a concurrent design in order speed up the disambiguation process. First of all, we branch out all entities, and start the disambiguation process on all of them in parallel. Secondly, we start the scoring process for each candidate for each entity in parallel as well (see Section 4.4.3). These steps improved our execution time tremendously. How much these steps improve the efficiency will of course depend on the machine used, and its architecture.

These parallelization steps do however have one weakness with the current implementation, namely that we end up calculating the scores twice for each candidate. This is because each entity operate by their own, independently of the others. Consider two neighboring entities; *A* and *B*. Both entities' candidates' scores are calculated independently, meaning that candidates of entity *A* calculates their similarity with candidates of entity *B*, while the candidates of entity *B* do the same with the candidates of entity *A*.

The optimal solution would obviously be to perform the calculations only once, allowing both sides to use the same result, rather than having both sides calculating the same thing simultaneously. Even though we currently have this disadvantage, parallelization is still of considerably more help than harm regarding efficiency. Fixing this problem would however improve the efficiency further.

6.5 Research Questions Revisited

We evaluate the work by revisiting the research questions introduced in Section 1.2, and discuss our findings in hindsight.

RQ: **How can we implement an entity linking system that selects the most relevant candidate article, taking entity ambiguity into account?**

In Section 3.2 we analyzed several state-of-the-art systems, and their approaches. We learned that they mainly emphasize data stored in a knowledge base in order to decide for the correct article when the entities are ambiguous.

We implemented a new entity linking system, using our own approach to the problem on the basis of the conducted research. We used data from a structured knowledge base in order to detect relations between entities in the input text, and selected candidates who seemed similar to each other for each entity. Our approach is described in detail in Chapter 4.

RQ1: **What kind of data and methods can be used to disambiguate ambiguous entities?**

The simplest method to decide for an entity candidate, is to select the most common candidate, meaning we will get correct result in many cases, but certainly not all. This is not a satisfying approach, because of all the cases we simply know we will fail to annotate correctly.

Instead, one might retrieve the entities from a structured knowledge base, and find similarities between them. These similarities could be defined by common in/out-links, common categories, etc. This approach of measuring similarities between entities assumes that similar entities usually are mentioned close to each other. Thus, the most similar entities should be chosen across all entity mentions.

RQ2: **How do today's entity linking systems select what candidate article should be used for each entity?**

Several modern state-of-the-art systems were analyzed in Section 3.2, showing that they retrieve information from a structured knowledge base, such as DBpedia, YAGO, or Wikidata. Even though the systems have minor differences, their approach is to discover relations and similarities across different entity mentions, usually in a similar manner to one of the methods mentioned for RQ1 above.

RQ3: **What is the best method for selecting correct article for an entity, and how can we maximize the efficiency for such a system?**

There is not necessarily one method that is superior to all others. Some systems specialize in annotating short texts, while others perform better for longer texts. Which approach is best will heavily depend on how the system will be used. What is certain however, is that a system should consider the context around every ambiguous entity mention, usually based on surrounding entity mentions, in order to decide which is the best candidate.

Regarding efficiency, it is beneficial to minimize the initial set of potential candidates, or at least minimize it early in the process by removing candidates considered unlikely to be correct anyway. Additionally, an awareness of which processes are possible and beneficial to do in parallel is a big advantage, as concurrency can speed up the program execution considerably.

Chapter 7

Summary

7.1 Conclusion

The entity linking task poses challenges in several research fields, which we have investigated in this thesis. On the basis of this research, we have designed and implemented an entity linking system from scratch, evaluated it, and compared its performance with other state-of-the-art entity linking systems.

We have used DBpedia in order to populate and disambiguate the entities within an input text. First, we found all entities in the input text, and represented them with a list of possible DBpedia articles about them, which we call *candidates*. Next, we used data from DBpedia to look for relations and similarities between different candidates across the entities considering the context, which consisted of nearby entities. We then selected the candidates we believed were similar to each other for each entity.

As we evaluated our system, we soon discovered some challenges with the entity extraction task. We did not end up with the ideal entity set as often as we would prefer. Wrongly identified entities propagated errors further down the line, as this also affected the context for the entities we actually extracted correctly. Regarding our solution for the entity disambiguation task, its effectiveness was satisfying considering our challenges with entity extraction, even though it was not as efficient as we anticipated.

It is also worth mentioning that there were some aspects of DBpedia which made the disambiguation process harder than it had be. For instance, DBpedia turned out to be more inconsistent than we initially thought, which obviously had a negative influence on our results. These inconsistencies includes the fact that similar entities do not always have the same kind of subjects, e.g., some countries lack a subject telling which continent it belongs to, which is present for the vast majority of countries.

We experienced that the WIKI-ANNOT30 dataset was not perfectly designed for our system, or vice versa. Ideally, we would like to use a dataset without any text fragments consisting of as little as down to two entities. Considering our context consists of four entities, we would prefer that each text fragment has at least five entities.

Experiments showed that our system was not as good as the systems we used as reference, such as *Wikify!* and *TagMe* when using the WIKI-ANNOT30 dataset. Our precision score showed that we managed to correctly annotate 45.0% of the entities we found, and our recall score showed that we managed to correctly annotate 28.8% of the entities according to the gold standard. This gives an F score of 35.1%.

Through experiments, we also found that each entity on average has about 17 potential candidates in DBpedia initially, meaning we have approximately 6% chance of selecting the correct candidate by random. Thus, we have certainly done something right, considering we achieved a 45% hit rate. however, there is still a way to go before our solution is effective enough to compete with the best entity linking systems out there.

Experiments have showed that candidate ranking methods based on machine learning often outperforms those using plain classification methods [42]. Thus, the future will probably bring more focus on developing sophisticated and intelligent systems using machine learning in order to disambiguate entities.

7.2 Future Work

There are multiple aspects that may be worth some more exploration, and serves as reasonable starting points for any future work.

- DBpedia is not a perfect knowledge base, which led to several problems for our disambiguation process. It could be beneficial to explore the possibility of replacing DBpedia with a knowledge base with more strict rules for how data is added, rather than just being a mirror of Wikipedia which anyone can edit, like DBpedia is. Even though we initially explored the possibility of using other knowledge bases as well, including the likes of Wikidata and YAGO, we believe it would be a good idea to revisit this decision once again.

- In order to resolve our problems with extracting entities from text, it might be preferable to use multiple named-entity extraction frameworks. Thereby, we can combine their results in order to find the outcomes it seems to be most agreement of between the different parsers.

- Our system's efficiency could be enhanced by avoiding multiple candidates computing the same calculations simultaneously, which is the case with the current implementation.

- We experienced that common words among the subjects sometimes had a large impact as we calculated the similarities between two candidates. This could possibly be avoided by using a discount factor for words, or string sequences, that have already been encountered.

- An approach where we consider multiple entities as one selection of multiple candidates at once might be favorable over our current approach. Currently, we pick the one candidate we think is most probable at any given time. However, in situations where we do not have one very superior candidate, it might be better to focus on which multiple candidates across the entities are most probable to occur together. This obviously complicates the process a bit, but will probably give a better reflection of the context and increase our chance of selecting the correct candidates, and is thus worth further investigation.

- If we enter a subject's page in DBpedia, we are presented with more information about the given subject. Especially the *"broader"* attribute which is present for some subjects may be of interest in our case. If we visit the *"Australia"* subject connected to the page of Australia, the *"broader"* attribute lists elements such as *"Countries in Oceania"*, *"Former British colonies"*, etc. By involving these subjects as well as the original, we can potentially avoid the problem we mentioned earlier, where the page for Australia does not mention *"Oceania"* among its primary subjects. However, it is worth noting that this would magnify the amount of processing needed in order to disambiguate entities.

Appendix A

Implementation

Our implementation use two classes: *Entity* and *Candidate* (see Figure A.1). The Entity objects represents the entity mentions in the text, while the Candidate objects represents DBpedia resources. Additionally, we have some separate files with code responsible for populating objects, parsing evaluation dataset, and organizing the general program flow. We present a short description of the Entity and Candidate classes.

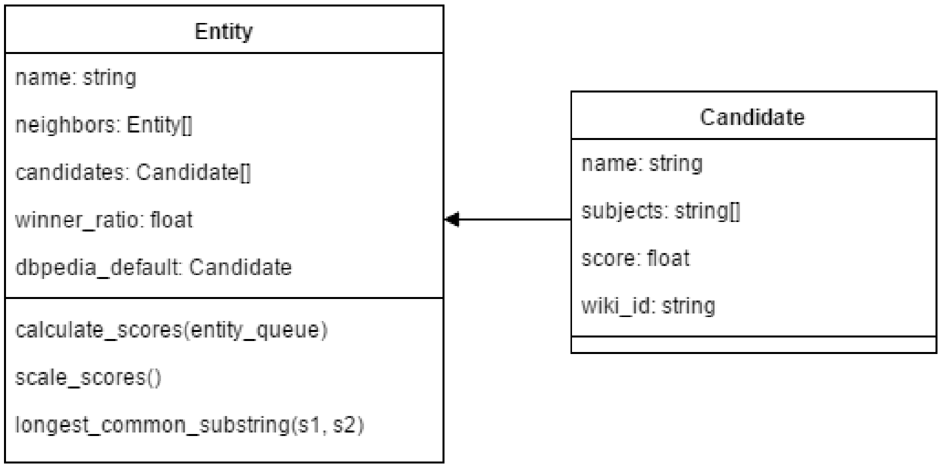


Figure A.1: Class diagram of our program.

Entity Class

An Entity object has the following set of variables:

name: A string value equivalent to the entity mention in the input text.

neighbors: A list of all entities defined as neighbors of the actual entity. All elements are other Entity objects.

candidates: A list of all resources the entity could represent in DBpedia. All elements are Candidate objects.

winner_ratio: A float value telling how superior the best candidate is. The value is set to zero if we have only one candidate, otherwise:

$$winner\_ratio = \frac{score\ of\ best\ candidate}{score\ of\ 2nd\ best\ candidate}$$

dbpedia_default: A copy of the Candidate object used as default value for this entity mention by DBpedia. Initially, the list of all candidates will also contain this object.

And the following set of methods:

calculate_scores(entity_queue): This method recomputes the score for the given candidate. Takes a *Queue* object used for concurrency as input. When the new scores are calculated, the whole Entity object is stored to the *entity_queue*.

scale_scores(): This method ensures that all the candidates' scores sums up to 1, and still keep their weights relative to each other. This way, their score can be seen as the probability of them being the correct candidate.

longest_common_substring(s1, s2): Takes two strings as input, and returns a score for how similar they are, based on their longest common substring.

Candidate Class

A Candidate object has the following set of variables:

name: A string value equivalent to the name of the DBpedia resource it represents.

subjects: A list of subjects connected to this candidate. The subjects are represented by strings.

score: Probability of this candidate being the correct one in its given context.

wiki_id: Unique ID of the Wikipedia article representing the actual candidate.

Bibliography

[1] Krisztian Balog, Heri Ramampiaro, Naimdjon Takhirov, and Kjetil Nørvåg. Multi-step classification approaches to cumulative citation recommendation. In *Proceedings of the 10th Conference on Open Research Areas in Information Retrieval*, pages 121–128. LE CENTRE DE HAUTES ETUDES INTERNATIONALES D'INFORMATIQUE DOCUMENTAIRE, 2013.

[2] Tim Berners-Lee, James Hendler, Ora Lassila, et al. The semantic web. *Scientific american*, 284(5):28–37, 2001.

[3] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python.* " O'Reilly Media, Inc.", 2009.

[4] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1247–1250. AcM, 2008.

[5] Kurt Bollacker, Patrick Tufts, Tomi Pierce, and Robert Cook. A platform for scalable, collaborative, structured information integration. In *Intl. Workshop on Information Integration on the Web (IIWeb'07)*, 2007.

[6] Ignacio Cano, Sameer Singh, and Carlos Guestrin. Distributed nonparametric representations for vital filtering: Uw at trec kba 2014. Technical report, DTIC Document, 2014.

[7] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to algorithms.* MIT press, 3rd edition, 2009.

[8] Joachim Daiber, Max Jakob, Chris Hokamp, and Pablo N. Mendes. Improving efficiency and accuracy in multilingual entity extraction. In *Proceedings of the 9th International Conference on Semantic Systems (I-Semantics)*, 2013.

[9] Milan Dojchinovski and Tomáš Kliegr. Entityclassifier. eu: real-time classification of entities in text with wikipedia. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 654–658. Springer, 2013.

[10] Joe Ellis, Jeremy Getman, Dana Fore, Neil Kuster, Zhiyi Song, Ann Bies, and Stephanie Strassel. Overview of linguistic resources for the tac kbp 2015 evaluations: Methodologies and results. In *Proc. Text Analysis Conference (TAC2015)*, 2015.

[11] Michael Färber, Basil Ell, Carsten Menne, and Achim Rettinger. A comparative survey of dbpedia, freebase, opencyc, wikidata, and yago. *Semantic Web Journal, July*, 2015.

[12] Lee Feigenbaum, Ivan Herman, Tonya Hongsermeier, Eric Neumann, and Susie Stephens. The semantic web in action. *Scientific American*, 297(6):90–97, 2007.

[13] Paolo Ferragina and Ugo Scaiella. Fast and accurate annotation of short texts with wikipedia pages. *arXiv preprint arXiv:1006.3498*, 2010.

[14] Paolo Ferragina and Ugo Scaiella. 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*, pages 1625–1628. ACM, 2010.

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

[16] John R Frank, Max Kleiman-Weiner, Daniel A Roberts, Ellen M Voorhees, and Ian Soboroff. Evaluating stream filtering for entity profile updates in trec 2012, 2013, and 2014. In *TREC*, 2014.

[17] Faegheh Hasibi, Krisztian Balog, and Svein Erik Bratsberg. On the reproducibility of the tagme entity linking system. In *European Conference on Information Retrieval*, pages 436–449. Springer, 2016.

[18] Benjamin Heinzerling, Alex Judea, and Michael Strube. Hits at tac kbp 2015: Entity discovery and linking, and event nugget detection. In *Proc. Text Analysis Conference (TAC2015)*, 2015.

[19] 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 Conference on Empirical Methods in Natural Language Processing*, pages 782–792. Association for Computational Linguistics, 2011.

[20] Yu Hong, Di Lu, Dian Yu, Xiaoman Pan, Xiaobin Wang, Yadong Chen, Lifu Huang, and Heng Ji. Rpi blender tac-kbp2015 system description. In *Proc. Text Analysis Conference (TAC2015)*, 2015.

[21] Heng Ji, Joel Nothman, and Ben Hachey. Overview of tac-kbp2014 entity discovery and linking tasks. In *Proc. Text Analysis Conference (TAC2014)*, 2014.

[22] Heng Ji, Joel Nothman, Ben Hachey, and Radu Florian. Overview of tac-kbp2015 tri-lingual entity discovery and linking. In *Text Analysis Conference*, 2015.

[23] Jingtian Jiang, Chin-Yew Lin, and Yong Rui. Msr kmg at trec 2014 kba track vital filtering task. *The Twenty-Third TREC Proceedings*, 2014.

[24] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick van Kleef, Sören Auer, et al. Dbpedia–a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web*, 6(2):167–195, 2015.

[25] Farzaneh Mahdisoltani, Joanna Biega, and Fabian Suchanek. Yago3: A knowledge base from multilingual wikipedias. In *7th Biennial Conference on Innovative Data Systems Research*. CIDR Conference, 2014.

[26] Pablo N Mendes, Max Jakob, Andrés García-Silva, and Christian Bizer. Dbpedia spotlight: shedding light on the web of documents. In *Proceedings of the 7th international conference on semantic systems*, pages 1–8. ACM, 2011.

[27] Rada Mihalcea and Andras Csomai. Wikify!: linking documents to encyclopedic knowledge. In *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, pages 233–242. ACM, 2007.

[28] David Milne and Ian H Witten. Learning to link with wikipedia. In *Proceedings of the 17th ACM conference on Information and knowledge management*, pages 509–518. ACM, 2008.

[29] Andrea Moro, Francesco Cecconi, and Roberto Navigli. Multilingual word sense disambiguation and entity linking for everybody. In *Proceedings of the 2014 International Conference on Posters & Demonstrations Track-Volume 1272*, pages 25–28. CEUR-WS. org, 2014.

[30] Andrea Moro, Alessandro Raganato, and Roberto Navigli. Entity linking meets word sense disambiguation: a unified approach. *Transactions of the Association for Computational Linguistics*, 2:231–244, 2014.

[31] Mohamed Morsey, Jens Lehmann, Sören Auer, Claus Stadler, and Sebastian Hellmann. Dbpedia and the live extraction of structured data from wikipedia. *Program*, 46(2):157–181, 2012.

[32] David Nadeau and Satoshi Sekine. A survey of named entity recognition and classification. *Lingvisticae Investigationes*, 30(1):3–26, 2007.

[33] Roberto Navigli and Simone Paolo Ponzetto. Babelnet: The automatic construction, evaluation and application of a wide-coverage multilingual semantic network. *Artificial Intelligence*, 193:217–250, 2012.

[34] Yuanyuan Qi, Ye Xu, Dongxu Zhang, and Weiran Xu. Bupt_pris at trec 2014 knowledge base acceleration track. Technical report, DTIC Document, 2014.

[35] 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, 2015.

[36] Avirup Sil, Georgiana Dinu, and Radu Florian. The ibm systems for trilingual entity discovery and linking at tac 2015. In *Proceedings of the Eighth Text Analysis Conference (TAC2015)*, 2015.

[37] René Speck and Axel-Cyrille Ngonga Ngomo. Ensemble learning for named entity recognition. In *International Semantic Web Conference*, pages 519–534. Springer, 2014.

[38] Fabian M Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: a core of semantic knowledge. In *Proceedings of the 16th international conference on World Wide Web*, pages 697–706. ACM, 2007.

[39] Ricardo Usbeck, Axel-Cyrille Ngonga Ngomo, Michael Röder, Daniel Gerber, Sandro Athaide Coelho, Sören Auer, and Andreas Both. Agdistis-agnostic disambiguation of named entities using linked open data. In *ECAI*, pages 1113–1114. Citeseer, 2014.

[40] Ricardo Usbeck, Axel-Cyrille Ngonga Ngomo, Michael Röder, Daniel Gerber, Sandro Athaide Coelho, Sören Auer, and Andreas Both. Agdistis-graph-based disambiguation of named entities using linked data. In *International Semantic Web Conference*, pages 457–471. Springer, 2014.

[41] Denny Vrandečić and Markus Krötzsch. Wikidata: a free collaborative knowledgebase. *Communications of the ACM*, 57(10):78–85, 2014.

[42] Zhicheng Zheng, Fangtao Li, Minlie Huang, and Xiaoyan Zhu. Learning to link entities with knowledge base. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 483–491. Association for Computational Linguistics, 2010.