Master's Thesis

Named Entity Recognition and Disambiguation with Wikidata on Arbitrary English Text

---

Yi-Chun Lin

Examiner: Prof. Dr. Hannah Bast
Adviser: Prof. Dr. Hannah Bast

Albert-Ludwigs-University Freiburg
Faculty of Engineering
Department of Computer Science
Chair of Algorithms and Data Structures

August 23th, 2021

Writing period

23. 02. 2021 – 23. 08. 2021

Examiner

Prof. Dr. Hannah Bast

Second Examiner

Dr. Fang Wei-Kleiner

Adviser

Prof. Dr. Hannah Bast

Declaration

I hereby declare, that I am the sole author and composer of my thesis and that no other sources or learning aids, other than those listed, have been used. Furthermore, I declare that I have acknowledged the work of others by providing detailed references of said work.
I hereby also declare, that my Thesis has not been prepared for another examination or assignment, either wholly or excerpts thereof.

Place, Date	Signature

Abstract

Named entity recognition and disambiguation (NERD) is the task of identifying and determining the meaning of named entities in a given text, for example, to tell whether the word "Amazon" is referred to the company or the river. This is done by linking the text spans to their corresponding entities in a knowledge base. NERD is an essential technique in many NLP applications including recommendation system and question-answering system, as it helps to extract and understand information from plain text. The task is however challenging, because the same piece of text can refer to different entities and the same entity can be referred to in multiple ways. In addition, the variety of texts – length, topic, style – also increases the difficulty of recognition and disambiguation. In this thesis, we propose an algorithm with configurable features for the task of NERD with Wikidata. In the recognition stage, we examine all possible text spans in the pre-generated named entity index, while reducing the complexity to linear scale by utilizing POS tags as a filter. In the disambiguation stage, we consider the popularity of each candidate entity and its similarity to the context. On top of the base model, we give the algorithm more flexibility in recognition by expanding the synonyms of certain named entities, and leverage Wikipedia abstracts to enrich the knowledge base. We further discard unlikely named entities to decrease false positives. In evaluation, we analyze the contribution of each component as well as the entire NERD system. Results show that synonym expansion and false positive reduction are very effective. In addition, our algorithm performs better and is capable of outputting more named entities on one dataset with shorter documents. On the other dataset consisting of news articles with rich context, our performance is also acceptable in comparison with the other sophisticated system.

Contents

List of Tables

List of Algorithms

1 Introduction

Named entity recognition and disambiguation (NERD), also referred to as entity linking (EL), is the task of identifying named entities in plain text and determining their meanings. Take the sentence "Amazon is founded by Jeff Bezos." for example. First of all, the task of named entity recognition (NER) is to point out the text span of named entities, which are "Amazon" and "Jeff Bezos". Then, the task of named entity disambiguation (NED) is to link the text span of each named entity to the correct entry in a knowledge base. It should have the ability to tell that "Amazon" is referred to the company, not the river or the rain forest. If we choose Wikidata as the target knowledge base, the output of NED would be:

- "Amazon": https://www.wikidata.org/wiki/Q3884
- "Jeff Bezos": https://www.wikidata.org/wiki/Q312556

1.1 Motivation

NERD is an essential technique for many applications in natural language processing, including recommendation system and question-answering system. It provides the ability to automatically extract information and understand the plain text. However, the task is challenging due to the ambiguity in nature language.

First, the same piece of text can refer to different entities. For example, the word "Amazon" could mean the company or the river depending on the context. On the other hand, the same entity can be referred to in multiple ways. One category is synonym, like the country “United States of America” can also be referred to as "U.S." or "USA". The other category is partial mention, which is common in certain content like news or sports. For example, use "Merkel" to refer to "Angela Merkel", or "Freiburg" to refer to "F.C. Freiburg" in text, especially when the term appears more than one time.

In addition, although we disambiguate by finding the corresponding entry in a knowledge base, not every entry in the knowledge base is a named entity, and vice

versa. Consider the text "an American company". The word "company" does have an entry in Wikidata, however it is only a simple noun. The word "American" does not have an entry in Wikidata, but is highly related the the entity "America".

Furthermore, the variety of texts also increases the difficulty of recognition and disambiguation. For example, news articles, Tweets and blog posts are all different in lengths and writing styles. Algorithms often need to be adjusted or trained according to different text characteristics.

1.2 Contribution

In this thesis, we propose an algorithm with configurable features for the task of NERD with Wikidata. It consists of two stages: named entity recognition (NER) and named entity disambiguation (NED).

In NER, we aim to identify as many named entities as possible. Ideally, this can be done by comparing all possible text spans with the pre-generated named entity index. Practically, we utilize POS-tag filter to reduce the examinations needed and achieve a linear complexity. To handle the issue of partial mention, we enhance matching flexibility by adopting the Wikidata property "family name". We also utilize the property "Demonym" – the natives or inhabitants of a particular country, state, city, etc – to recognize terms like American.

In NED, we consider the popularity of each candidate entity and its similarity to the context. We measure the similarity by the overlaps between the context and the description of the candidates. Also, we leverage the information from Wikipedia abstracts to make better decisions in disambiguation. Finally, we discard unlikely named entities to decrease false positives by conditionally removing consequent single-word named entities.

We generate two benchmarks for evaluation. One dataset consists of shorter documents with diverse topics. The other dataset consists of news articles. We conduct experiments on various versions of our algorithm, using the designed configurable framework. We also compare with one state-of-the-art system. Results show that the use of the property "Demonym" and the false positive reduction are very effective. In addition, on the dataset with shorter documents, our algorithm performs better and is able to correctly recognize and disambiguate more named entities.

1.3 Structure of the Thesis

The thesis is structured in the following way. In Chapter 2 we review some related works and summarize the lessons learned from them. In Chapter 3 we present the base model of our NERD algorithm and introduce five features on top of it. In Chapter 4 we describe the experiment setup and discuss the evaluation results on each feature as well as the entire NERD system. We conclude in Chapter 5.

2 Related Work

Wikify! [1] is the first work that introduces the NERD system. Followed by Cucerzan's work [2] and Milne and Witten's work [3], these earlier approaches aim to bridge information from knowledge bases to ordinary web pages. They detect the most popular or important entities and disambiguate them with Wikipedia. Since the recognition process is rather conservative, many entities in the text are not detected. Their decision of focusing on only certain entities results in higher precision rates but lower recall rates.

Hoffart et al. [4] focus on NED and propose an integrated algorithm considering the popularity of a candidate, the similarity between a candidate and the context, and the coherence of all entities in the text. The similarity measure is based on a offline data-mining step that determines the keyphrases for each entity in the knowledge base. The consideration of coherence, inspired by Kulkarni et al. [5], is under the assumption that the query text is not too short thus has sufficient number of named entities and these named entities focus on the same topic. They model the NED problem as a weighted graph of mentions and candidates, in which the similarity and coherence are represented by the corresponding edge weights. It is solved globally by computing a sub-graph that contains all mention nodes and one candidate-mention edge for each mention. Together with Stanford NER tagger [6] to identify mentions and YAGO2 [7] knowledge base for candidate generation, they reach a good result in the AIDA CoNLL-YAGO dataset, which is also generated by them.

Piccinno and Ferragina propose WAT [8], an enhanced version of their previous work TagMe [9]. WAT consists of three stages: spotting, NED and pruning. This provides the possibility to spot more mentions in the beginning and prune the inconsistent ones in the last stage. The spotting stage detects possible mentions and generates candidate lists based on an offline pre-processed entity index from Wikipedia. The NED stage offers two types of algorithm. The vote-based algorithms disambiguate locally considering the relevance between entities within a context window and the popularity of each candidate. The graph-based algorithms are also mention-candidate graph as Hoffart et al. [4], but with different kinds of relevance

measures and optimization algorithms. They reach good results in many evaluations in other recent works. In their own evaluation on ERD dataset, they analyze the contribution of each stage as well as the entire NERD system. Though the graph-based algorithms perform better than vote-based ones in the NED-only test, the performance have little difference in the NERD test, in which the noise introduced by the spotting stage largely degrades the performance of NED. They therefore point out the importance of the spotting and pruning stage as these two stages are responsible for many false-positives.

Kolitsas et al. [10] propose a neural-network system to jointly solve the NER and NED problem. They provide all possible spans with their candidate lists as input data. The best candidate is chosen by the probability model, the context-aware similarity and optionally the global coherence. The network is trained with the golden mention-entity pairs as positive cases and the other mention-candidate pairs as well as the wrong spans as negative cases, thus has the ability to jointly determine spans and choose candidates. They generate candidate lists based on an offline-generated probabilistic entity index built by Ganea and Hofmann [11], collecting information from Wikipedia hyperlinks, Crosswikis and YAGO dictionaries. They compute similarity between context-aware word embeddings of the span, obtained by bidirectional LSTM, and the pre-trained entity embeddings by Ganea and Hofmann [11]. In evaluation, they significantly outperform other NERD systems on AIDA CoNLL-YAGO dataset, which is in the same domain of their training data. For datasets out of their training data, their NED model with the Stanford NER tagger [6] performs better. The enforcement of global coherent boosts up the performance of in-domain datasets while drops the performance in out-of-domain ones.

We learn some lessons from these works. First, the components of NED algorithms are basically "probability", "similarity", and "coherence". Among them, "coherence" deserves careful handling. Since the errors may dominate the result when we force all entities in the documents to be coherent, the rule should be applied only when we have a high confidence on the algorithm. Second, offline data-mining on entities is essential. It can be used to generate an entity index with popularity or probability information. It can also provides keyphrases or embeddings of the entity for later comparison. Finally, in the trade-off between precision and recall rate, more false-positives are introduced if we want to recognize more entities. To prevent error propagation, there should be a way to handle the false-positives generated by NER, e.g. by a post-processing stage, or by jointly solving NER and NED.

3 Approach

In this chapter, we present the proposed NERD algorithm. The basic flow of named entity recognition and disambiguation are shown in Section 3.1 and Section 3.2. In Section 3.3, five configurable features are introduced.

3.1 Named Entity Recognition

The task of named entity recognition (NER) is to locate the text span of named entities in given text. A basic approach is to use the part-of-speech (POS) tagging, which is the process of determining the grammatical category of each word in the sentence. The Penn Treebank tagsets [12] is a widely used tagset when it comes to POS tagging. You can find the description of some common tags mentioned in this thesis in Table 1.

Tag	Description
NNP	Proper noun, singular
NNPS	Proper noun, plural
NN	Noun, singular or mass
NNS	Noun, plural
VBD	Verb, past tense
VBN	Verb, past participle
IN	Preposition or subordinating conjunction
JJ	Adjective

Table 1: A subset of the Penn Treebank tagset.

Let's take the following sentence as an example to see how POS tagging looks like and how it relates to NER. The POS tag of each word is shown in grey background right after the word. The named entities are in bold.

Amazon `NNP` was `VBD` founded `VBN` by `IN` **Jeff** `NNP` **Bezos** `NNP` . `.`

In the sentence, "Amazon" is a proper name, with the tag `NNP`; "was" and "founded" are some type of verbs; "by" is a preposition; "Jeff" and "Bezos" are again

proper names with the tag `NNP`. It's obvious that "Amazon" and "Jeff Bezos" are named entities, and they all have the POS tag `NNP`. That is, we can easily marking all words that has the `NNP` tag as named entities. However, there are two problems if we do NER by POS-tagging. First, we don't know whether adjacent words belong to the same named entity. Second, words in a named entity do not always have the tag `NNP`. In the phrase shown below, if we mark all `NNP` words, we will get two named entities: "United States" and "America". We miss the entire "United States of America".

United `NNP` **States** `NNP` **of** `IN` **America** `NNP`

In order to recognize more named entities, especially those cannot be fully detected by POS-tagging, the ultimate way is to compare all of the possible text spans in the given text with a named entity index generated from knowledge bases. Take "United States of America" for example, we need to compare each of the following text spans with the named entity index:

- United
- United States
- United States of
- United States of America
- States
- States of
- States of America
- of
- of America
- America

In this way, we do not miss any possibility and can recognize the entire "United States of America". However, this process could be very time-consuming when the input text goes longer. Because it has an order $O(k^2)$ given a text of length k. Fortunately, the number of comparison can be largely reduced. In Section 3.1.1 and Section 3.1.2, we propose two improvements to reduce the number of comparison, namely POS tag filter and named entity index.

3.1.1 POS Tag Filter

The first improvement to speed up the recognition process is to use the POS tag as a filter to reduce the number of comparison needed. The function of the POS tag filter is to do a quick scan to the given text to spot the position of possible named entities. Since most of the words in the given text are not named entities, e.g. "was", "founded", there is no need to further check the text spans starting with these words. Due to the fact that whether a word is (part of) a possible named entity could be roughly indicated by its POS tag, the idea is to only check the words that have *possible* POS tags, and skip the words that have other POS tags.

We prefer a loose filter than a strict filter. With strict filter policies, we may miss out some possible named entities. This kind of false-filtering is not what we want as our goal is to recognize more named entities. To this end, we consider not only `NNP` but also `NN` as our candidate tags. Particularly, a word having tag `NNP`, `NNPS`, `NN` or `NNS` is possible to be in a part of a named entity and requires further examination. Meanwhile, when filtering a word, not only the tag of the current word but also the tag of the next word are taken into consideration. For example, consider named entities like "My Chemical Romance" (an American punk band) or "My Neighbor Totoro" (a Japanese anime film), both of them have POS tags of the form `PRP$`, `NNP`, `NNP`. If we only check the POS tag of the current word, we will skip the word "My" and then fail to detect the entire name entity.

Algorithm 1 POS Tag Filtering

```
valid_tags ← { "NNP", "NNPS", "NN", "NNS" }
w_p ← the word at position p in the query
function IsPossibleNE(w_p)
    tag_p ← PosTag(w_p)
    tag_{p+1} ← PosTag(w_{p+1})
    if tag_p ∈ valid_tags or tag_{p+1} ∈ valid_tags then
        return true
    else
        return false
    end if
end function
```

Algorithm 1 states the proposed POS tag filtering. Given a word in the query, we determine if it is possible to be (part of) a named entity by the POS tag of the word itself and the word after it. For the function PosTag, we use spaCy [13] as the POS tagger, because it is one of the stat-of-the-art tagger and performs fast. In summary,

by utilizing the POS tag as a filter, the amount of words needing further examination can be largely reduced. Thus, the comparison needed becomes to a linear scale $O(k)$, given a text of length k.

3.1.2 Named Entity Index

The second improvement to speed up the recognition process is to create a named entity index in advance. The index stores all the named entities from the knowledge base according to the beginning word and the length of named entities. While the POS tag filter reduces the number of words needing examination, the aim of the named entity index is to reduce the examination time of each word.

Consider the first word "Amazon" in the example "Amazon was founded by Jeff Bezos". Since "Amazon" has the tag `NNP`, we need to check itself as well as all the phrases starting with it until the end of the sentence, i.e.

- "Amazon"
- "Amazon was"
- "Amazon was founded"
- ...
- "Amazon was founded by Jeff Bezos"

together 6 comparisons needed to be made. However, the entities starting with the keyword "Amazon" in the named entity index, like "Amazon", "Amazon Kindle", "Amazon rainforest" and so on, are finite. Let's say the named entities starting with "Amazon" are of length 1, 2, and 3. Then, we don't need to compare a six-word phrase "Amazon was founded by Jeff Bezos" or other lengths that are not exist in the named entity index. By knowing this fact, only 3 comparisons are needed. Therefore, if we build a named entity index in advance, such that we can easily know the possible lengths of named entities given a starting word, we can further reduce the number of following phrases a word needs to be checked.

We build up the named entity index based on the information in Wikidata. Particularly, we use the "name" and "synonyms" property of an entity as the key, and the "QID" property as the value in the index. Each key is stored under a hierarchical structure: the first layer is according to its starting word, while the second layer is according to its length. The value is an array of QIDs, served as a list of candidate entities that could refer to the key.

QID	Name	Synonyms
Q30	United States of America	USA; United States
Q145	United Kingdom	
Q9212	United States Army	USA

Table 2: An example knowledge base to demonstrate how to build the named entity index.

Given a toy knowledge base with only three items in Table 2, the pre-established named entity index is shown in List 3.1. First, we have "United States of America". We store it under "United" and then "4" to indicate its beginning word and length. Its value is its QID, which is Q30. The synonyms, "USA" and "United States" are stored as the same manner. "USA" is under "USA" and then "1"; "United States" is under "United" and then "2". Both of them have the value Q30. Then, we have "United Kingdom". We store it under "United" and then "2" and with value Q145. Now there are two named entities under "United" and "2". Finally, we have "United States Army" and its synonym "USA". Since there is already an "USA" in the index, we add the QID Q9212 into the value array of the exist key to indicate that all these QIDs could point to the same key.

3.1.3 The Recognition Flow

We have introduced the POS tag filter to spot the possible named entity words, and the named entity index to speed up the examination of the words. Now let's look at the recognition flow in details. Given a query sentence, we check it word by word with the POS tag filter, from the beginning to the end. If we find a possible word, we compare the word and its following text spans with the named entity index to see if there is a match. Particularly, we only examine the spans of possible lengths, and choose the longest one if there are more than one match. The text span that is recognized as a named entity will not be examined again. Finally, we discard the text spans that do not include any `NNP` tag to avoid recognizing simple nouns. Algorithm 2 describes the process of examining a possible word in the named entity index. The formal description of the NER flow is in Algorithm 3.

Let's use the toy index in 3.1 as the named entity index and run the NER process on the text "the United States of America (USA)". The POS tag of each word is listed below.

Listing 3.1 An Example Named Entity Index

```
{
    "United": {
        "2": {
            "United States": [Q30],
            "United Kingdom": [Q145]
        },
        "3": {
            "United States Army": [Q9212]
        },
        "4": {
            "United States of America": [Q30]
        }
    },
    "USA": {
        "1": {
            "USA": [Q30, Q9212]
        }
    }
}
```

Algorithm 2 Examine a word in Named Entity Index

$NE\_index \leftarrow$ BuildIndex()
function EXAMINE(w_p)
  $pos\_lens \leftarrow$ possible lengths of named entities starting with w_p
  $\hat{l} \leftarrow 0$, keep track of the longest length
  foreach l in pos_lens **do**
    if $chunk_{p,l} \in NE\_index$ *and* $l > \hat{l}$ **then**
      update the longest match $\hat{l}$, $QIDs$
    end if
  end for
  return $\hat{l}$, $QIDs$
end function

Algorithm 3 Named Entity Recognition

$p \leftarrow 0$, start from the beginning word of the query
while w_p is not NULL **do**
  if IsPossibleNE(w_p) **then**
    $\hat{l}$, $QIDs \leftarrow$ Examine(w_p)
    if $\hat{l} \neq 0$ **then**
      if $span_{p,\hat{l}}$ contains "NNP" **then**
        Output.insert($span_{p,\hat{l}}$, $QIDs$)
      end if
      $p \leftarrow p + \hat{l}$
      continue
    end if
  end if
  $p \leftarrow p + 1$
end while

the `DT` United `NNP` States `NNP` of `IN` America `NNP` (`-LRB-` USA `NNP`) `-RRB-`

The first word "the" does not have a possible tag, so we skip it. The second word "United" has the tag `NNP`, so we look up the index and find out that the possible lengths starting with "United" are 2, 3 and 4. Therefore, we extract the two-word phrase "United States", the three-word phrase "United States of" and the four-word phrase "United States of America" from the query and look up the index to see if any of them exists thus is a named entity. Since "United States" and "United States of America" exist in the index, we choose the longer one, "United States of America". Now we successfully recognize a named entity, with its QID array in hand for later usage in the NED stage. The next word to check is "(", which does not have a possible tag. Then is the word "USA", which has the tag `NNP`. So we look up the index again and the possible length is 1. The single word "USA" exist in the index, with the QID array consisting of two QIDs. Now we have recognized another named entity. We continue to the next word, which is the last word ")" and does not have a possible tag. The process is done. The recognized named entities as well as their candidate lists, namely "United States of America"([Q30]) and "USA"([Q30, Q9212]), are passed to the next stage.

3.2 Named Entity Disambiguation

The task of named entity disambiguation(NED) is to link the recognized named entity to the respective item in a knowledge base, which is Wikidata in this thesis. Therefore,

the task of NED is to determine the URL or the QID of each recognized named entity. Here, the recognized named entities come from the output of the previous stage, NER. Since a text span could refer to different items in a knowledge base, the NED algorithm needs to choose the most suitable item among the candidates. For example, there are at least two "Obama" in Wikidata. One is the former U.S. president; the other is a city in Japan. When "Obama" is recognized in a given text, NED needs to determine which "Obama" does it mean. A straight forward approach is to choose the most popular candidate. However, the drawback is easy to see. Since the most popular item with the name “Obama” is the president Obama, the approach always links “Obama” to the president, and fails in the case of “Obama is a city in Japan.” Therefore, more clues should be taken into consideration.

3.2.1 Context-Aware Disambiguation

When determining the meaning of a text span, the context of the query usually provides great hints. A candidate entity that is more related to the context is more possible to be the correct answer. This is also how human understands the meaning of texts. To measure which candidate is more related, we need to compare the context with each candidate. To do so, we have to define the following three concepts above all: 1) What is the "context" of a given text. 2) What can represent a "named entity". 3) How to measure the "similarity" between a "context" and a "named entity".

First, the context of a given text can be roughly represented by all the nouns in the text. Since our goal is to use the context to distinguish between different named entities, not to understand the actions or emotions in the text, we neglect verbs, adjectives or combinations between nouns and them. Particularly, we collects all the words in the query having POS tag `NNP`, `NNPS`, `NN` and `NNS` to represent the context of the query.

Then, to represent a named entity, we use the information in Wikidata. A straightforward idea is the information in its description. Particularly, we use all the words in the Wikidata property "name", "description" and "synonyms" of a named entity without further processing. Theoretically, we could remove the stop words or also do the POS-tagging and only consider the nouns. However, our final goal is to compare each named entity with the "context", which already contains only the nouns and proper nouns. Therefore, we choose to save the time and effort of further processing each named entity, as there might be a lot of them.

Finally, to measure the similarity between context and a named entity, the idea is to see if they have any keywords in common. Particularly, we count the number of

overlap words between the context and each named entity. The one with the most overlaps is the most related named entity.

Let's take the two sentences in Table 3 as example: we want to determine the meaning of the word "Obama" in both sentences. Assume that there are only two candidate "Obama" in Wikidata, as shown in Table 4. First, we need to extract the sentence contexts. In the first sentence, "Obama" and "US" have the tag `NNP`, and "president" has the tag `NN`. The three words are the context of the sentence. The context of the second sentence is of the same manner: "Obama" and "Japan" are `NNP`, and "city" is `NN`.

Sentence	Context
Obama was the president in US.	Obama, president, US
Obama is a city in Japan.	Obama, city, Japan

Table 3: An example to illustrate context-aware disambiguation. The context of the two example sentences.

QID	Name	Synonyms	Description
Q76	Barack Obama	Obama	44th president of the United States
Q41773	Obama		city in Fukui prefecture, Japan

Table 4: An example to illustrate context-aware disambiguation. The information of the two "Obama" entities in Wikidata

	Sentence 1	Sentence 2
Q76	Obama, president	Obama
Q41773	Obama	Obama, city, Japan

Table 5: An example to illustrate context-aware disambiguation. The overlap words between the context of the two sentences and the two candidate named entities.

Now we have the context and the information of the candidate named entities in hand. Let's disambiguate the "Obama" in both sentences by measuring which candidate named entity has more overlaps to the context. The overlaps between each

candidate and each sentence are shown in Table 5. For the first sentence, since there are more overlaps with the candidate Q76, we disambiguate the "Obama" to Q76, the president. Meanwhile, we disambiguate the "Obama" in the second sentence to Q41773, the city, as it has more overlaps with the context.

3.2.2 The Disambiguation Flow

The example above is well designed to demonstrate the concept of disambiguation by comparing overlaps. In real cases, there could be no overlaps at all, or the amount of overlaps may not be positively correlated. Therefore, we make the disambiguation considering not only the relevance of the candidate, but also its popularity. For a recognized named entity, given all its possible candidates, disambiguate by choosing the candidate with the highest score, where

$$score = popularity\_score + similarity\_score$$

The popularity score comes from the entity's property sitelinks in Wikidata. Popularity score is an integer. In the Wikidata we use, it is in the range from 0 to 367. A higher number of sitelinks indicates a more popular entity.

$$similarity\_score = n\_overlaps \times weight$$

The similarity score is proportional to the number of overlaps between the context and the entity. The weight controls how significant an overlap is. We design the weight from two aspects. First, the weight should have the ability to beat the popularity score. Denote the maximum popularity score as P_{max}. We choose the weight to be near the half of P_{max}. This means, assume there are two entities, one is the most popular, the other is the least popular, then our algorithm should choose the least popular one, if it has two more overlaps with the context. Second, the weight should be a function to the length of the context. In a longer query, the context may contain more words but be less representative. In this case, we should lower the significance of every overlap, i.e. lower the weight. In our implementation, the weight goes down to around one third of P_{max} when there are more than 10 words in the context.

$$weight = \begin{cases} P_{max}/3 & \text{, more than 10 words in the context} \\ P_{max}/2 & \text{, otherwise} \end{cases}$$

Theoretically, there is one more aspect we should take into consideration to design the weight. The weight should also be a function of the description length of each

named entity, as more words in the description provides a higher chance of overlaps. However, the description in Wikidata tend to be short: 96% of the description are less than 10 words. In this case, a fixed weight is sufficient. Note that the behavior changes when we later introduce the Wikipedia abstract in Section 3.3.4.

3.3 Configurable Features

We have introduced the base model of our named entity recognition and disambiguation algorithm. In this section, we try to further improve the correctness of recognition and disambiguation in the following aspects. In Section 3.3.1 and Section 3.3.2, we give the algorithm more flexibility in recognition by expanding the synonyms of certain named entities. In Section 3.3.3 and Section 3.3.4, we enrich the knowledge base so that more information can be accessed. In Section 3.3.5, we reduce the false-detected named entities. Each improvement is implemented as a configurable feature such that it could be turn on or off in the algorithm. Their effectiveness will be discussed in the evaluation section in Chapter 4.

3.3.1 Family Name

One common sort of error is related to the name of a person. Sometimes the query sentence does not state the complete name when mentioning a person, but only its last name. This may induce a recognition problem. For example, in the sentence "Armstrong was stripped of all seven Tour de France titles.", though we know it means "Lance Armstrong", the entity "Lance Armstrong" in Wikidata does not have its last name "Armstrong" as its synonym. That means, the key "Armstrong" in our named entity index does not have a value that links to "Lance Armstrong". This makes it impossible to disambiguate "Armstrong" correctly as the answer is not included in the candidate list. To solve this kind of error, we can simply add the last name “Armstrong” to the synonym list of "Lance Armstrong". In general, we can add all people’s last name to its synonym. Particularly, we use the property "type" and "family name" in Wikidata to implement this feature. If an entity is of type “person” and has the property “family name”, add its family name to its synonym.

3.3.2 Demonym

Consider the following sentence "Amazon is an American company.", the POS tags are shown below.

Amazon NNP is VBZ an DT American JJ company NN . .

"Amazon" is a named entity for sure, but what about "American"? By definition, it is an adjective thus not a named entity. However, it is strongly related to the named entity "America". If we can link "American" to "America", the computer can understand more about the text. This is where "demonym" plays a role. Demonym is a noun denoting the natives or inhabitants of a particular country, state, city, etc. It is an property in Wikidata. Therefore, we can utilize the information by adding all the words in demonym to the synonym of certain entities. In particular, if the entity is a country and has the property "demonym", add its demonym to its synonym. This is practical as the demonym itself may not be an entity in the knowledge base. Even the demonym is an entity, it is still good to link the demonym to its country for better understanding. Also note that if this feature is on, the *valid_tags* in Algorithm 1 should also include the tag `JJ`, such that the denonyms can pass our POS tag filter to do recognition and disambiguation.

3.3.3 Large Database

In the early stage of development, we use a condensed version of Wikidata, which excludes less popular entities. The advantage of the condensed version is that it is lighter thus it can save time and space when running the algorithm. The disadvantage is easy to see. The condensed version leads to recognition limits as not all entities are included. Therefore, it is reasonable to try the full version of Wikidata and to compare the performance difference.

3.3.4 Wikipedia Abstract

In NED stage, we compute the similarity score of a candidate using its description in Wikidata. But sometimes the description in Wikidata contains too little information and the disambiguation algorithm falls back to only depends on the popularity score. One possible improvement is to include more information to represent each named entity. We choose the abstract paragraph of the corresponding Wikipedia page to provide more informative details.

Consider the sentence "Armstrong was stripped of all seven Tour de France titles." Its context is ["Armstrong", "Tour", "France", "titles"]. Now we want to disambiguate the word "Armstrong". Assume there are only two candidate named entities listed in Table 6. Since both of the candidates have no overlaps with the context, the NED algorithm can only depend on their popularity scores and choose Neil Armstrong as

QID	Name	Description
Q1615	Neil Armstrong	American astronaut and the first human to walk on the moon
Q2172	Lance Armstrong	American cyclist

Table 6: The information of the two "Armstrong" entities in Wikidata.

it has higher popularity score. However, if we also check their Wikipedia abstract, we will find out that the term "Tour de France" appears in Lance Armstrong's Wikipedia abstract. The overlaps contribute to his similarity score and leads to the correct disambiguation result.

Note that the length of Wikipedia abstract differs in a large range. The longer the abstract, the higher chance there is an overlap with the context. Thus, we need to further adjust the weight to compensate the effect of different lengths. The weight is inversely proportional to the logarithm of the length such that longer abstract gets lower weight. Particularly, we put all the words of the property "name", "synonyms", "description" and "Wikipedia abstract" into an unordered set. Denote the length of the set as len, the adjusted weight would be

$$adjusted\_weight = \begin{cases} \frac{4}{log_2(len)} \times w & , len > 16 \\ w & , \text{otherwise} \end{cases}$$

3.3.5 NNP Reduction

One sort of error comes from the false-recognized named entities. That is, the NER stage recognizes certain text span as a named entity, but it is actually not. Specifically, we are looking at the pattern where the entire entity does not exist in Wikidata but each word of the entity exists. For example, "Bank Duta" is an Indonesia bank, which is a named entity but has no entry in Wikidata. Meanwhile, "Bank" (a film by Charlie Chaplin) and "Duta" (a family name) are both entities in Wikidata. Thus, the text "Bank Duta" results in two false-recognized named entities. In fact, it is less possible to have single-word named entities in sequence, especially when they are not so popular or related. Therefore, we prevent this kind of false-recognition by removing the recognition of consequent single-word named entities, except any of the named entity has a score larger than P_{max}. Because the score is the summation of popularity score and similarity score, we can use the score as a comprehensive index to indicate if an entity is very popular or related to the context.

4 Evaluation

In this chapter, we evaluate the proposed NERD algorithm on two datasets. We compare the performance of different configurations of our algorithm, as well as with one state-of-the-art system. Section 4.1 introduces the two datasets. Section 4.2 describes the metrics we evaluated on. In Section 4.3, we discuss the experimental results. Finally, we conduct an error analysis in Section 4.4.

4.1 Datasets

We use Wikidata as our knowledge base and conduct experiments in English. Since there are no established dataset with Wikidata annotations, we modify two exist datasets, ClueWeb12 FACC1 [14] and AIDA CoNLL-YAGO [4], by mapping their annotations to Wikidata QIDs.

ClueWeb12 FACC1 is introduced by Google. It consists of text from 456,498,584 English web pages with Freebase annotations. Due to the extremely large size of the data, the annotation process was automatic. In our benchmark, we randomly choose 50,000 texts from the dataset and use the Wikidata property "Freebase ID" to map the Freebase annotations to Wikidata annotations.

AIDA CoNLL-YAGO is introduced by Hoffart et al [4]. It is based on the CoNLL-2003 data set, which consists of 1,393 English and 909 German news articles with named entities annotated by categories (LOC, ORG, PER, or MISC). These named entities are further manually annotated by Hoffart et al. with YAGO2 entity name, Freebase ID, and Wikipedia URL annotations. In our benchmark, we use the 1,393 English news articles and map the annotations from either Freebase ID or Wikipedia URL to Wikidata annotations.

The statistics of the two benchmarks are shown in Table 7. ClueWeb contains text from a large size of webpages thus covers a wide range of topics. The automatic annotations are usually of high quality, but they may still contain errors. The average length of each document is 26 words with 1.55 named entities. On the other hand, AIDA focuses on news topics and has a smaller size of corpus. The manual

Dataset	Clueweb	AIDA
Topic	mixed	news
Annotation Type	automatic	manual
Num. Doc.	50000	1393
Num. N.E.	77412	34856
Num. InKB N.E.	77412	27507
Avg. Word/Doc.	26.16	217.48
Avg. InKB N.E./Doc.	1.55	19.79

Table 7: Benchmark Statistics. Doc. stands for document. N.E. stands for named entities.

annotations provide better quality. The document length is longer, averagely 217 words per document with 20 named entities.

4.2 Metrics

We report Micro F1 and Macro F1 scores of NERD, where

$$F1 = \frac{2 \times precision \times recall}{precision + recall}.$$

Precision is the ratio of correctly reported named entities among the algorithm outputs. Recall is the ratio of correctly reported named entities among the ground truth. By considering both precision and recall, we can avoid the situation of only reporting few named entities with high confidence to aim a high precision, or reporting every possibility of named entities to aim a high recall.

Micro F1 aggregates data from all documents to compute one score. On the other hand, Macro F1 computes one score per document and takes average over all documents to get the final score. When the characteristics of each document – number of named entities, topic and so on – are diverse, Macro F1 score is preferable.

We compute in the strong matching setting, where a "match" requires the algorithm output to have the exactly same boundary and annotation with the ground truth. In addition, since the annotations are originally in other knowledge bases, some entities in the ground truth do not have corresponding entities in Wikidata. We therefore focus on named entities in Wikidata and report the so-called *InKB* scores. For entities in the ground truth that do not have valid annotations in Wikidata, the metrics ignore the algorithm's output on such entities in score computation.

configuration	**Clueweb** Micro F1 Macro F1	**AIDA** Micro F1 Macro F1	**memory** (GB)
base	39.49 39.98	50.9 50.32	**3.80**
base + family name	39.62 40.48	53.47 52.29	3.89
base + demonym	41.78 42.74	**55.65** **56.34**	**3.80**
base + large database	39.86 40.71	51.08 50.68	5.19
base + Wikipedia abstract	38.92 39.29	51.03 50.13	5.86
base + NNP reduction	**47.06** **42.95**	54.26 53.22	**3.80**

Table 8: NERD results of single feature. Micro, Macro F1 scores and the memory usage are shown. We highlight the best result in bold and the second best with underline.

4.3 Results and Discussion

We test on our base NERD algorithm, as well as different combinations of the five features – *family name*, *demonym*, *large database*, *Wikipedia abstract*, *NNP reduction* – on top of the base model. Also, we compare to one of the state-of-the-art NERD system – AmbiverseNLU[1]. AmbiverseNLU uses KnowNER [15] for NER, and an enhanced version of AIDA [4] for NED. We use their Docker to conduct experiments.

4.3.1 Single Feature

First, we apply only one feature each time on the base model to see the effectiveness of each feature. The F1 scores on two benchamrks and the memory usage are shown in Table 8. The percentage change of false positive and false negative counts aggregated from two benchmarks are shown in Table 9. *NNP reduction* and *demonym* provide great improvements without extra memory usage. On the other hand, *large database* and *Wikipedia abstract* provide little improvements while consuming much more memory. Following are detailed discussion on each feature.

[1]https://github.com/ambiverse-nlu/ambiverse-nlu

configuration	false positive		false negative	
	counts	**% change**	**counts**	**% change**
base	89,305	-	53,194	-
base + family name	90,711	1.57%	51,745	-2.72%
base + demonym	89,066	-0.27%	48,696	**-8.46%**
base + large database	91,659	2.64%	52,056	-2.14%
base + Wikipedia abstract	89,943	0.71%	53,659	0.87%
base + NNP reduction	52,771	**-40.91%**	53,972	1.46%

Table 9: False positive and false negative counts of each feature

1. *Family name* uses little extra memory and have a good F1 score gain on AIDA. The gain on Clueweb dataset is not so significant. Since AIDA consists of news articles, it may contain more family names as well as related context thus benefit from this feature. Though *family name* helps to detect more named entities, it also brings new ambiguity by providing more candidates when disambiguating a name. Together, it reduces 2.72% of false negatives but also increases 1.57% of false positives.

2. *Demonym* is very effective on both datasets. It uses nearly no extra memory and provides a significant gain on AIDA and a good gain on Clueweb. Linking a country's demonym as its synonym makes sense and generates no new ambiguity as each country is distinct. This coincide with the result of reducing 8.46% of false negatives and introducing no new false positives.

3. *Large database* uses 1.4GB extra memory compared to the condense version of database. Surprisingly, it brings only minor improvements on both datasets. The full version of database do contain more information, but the benefit of recognizing more named entities is cancelled out by introducing more ambiguity. It results in a 2.14% reduction on false negatives and a 2.64% increment on false positives. We can think of the condensed version as a pre-prossessed version to remove noise. To make the best out of the full version of database, further consideration and disposal are required.

4. *Wikipedia abstract* uses about 2GB extra memory but actually degrades the performance a bit. Though the idea makes sense in our example sentence, it slightly increases both false positives and false negatives overall. The subtle changes means that this feature has almost no impact. It could due to the mechanism of our similarity measure – counting overlaps between query context

and candidate's description – is not suitable for a longer and less focused description.

5. *NNP reduction* is a very effective feature. It contributes large improvements on both datasets without extra memory usage. The improvement comes from the 40.9% reduction of false positives. Though it also brings 1.46% more false negatives, the benefits outweigh the disadvantages. The result supports our hypothesis that consequent single-word named entities are relatively rare.

4.3.2 Multiple Features

Then, we look at the performances of different combination of features. From the discussion above, we can enable *family name*, *demonym* and *NNP reduction* by default as they all provide positive effects and require no much extra memory. We denote *enhanced* as the version of base algorithm plus the three features. We further want to know the effects of *large database* and *Wikipedia abstract* on top of the enhanced version, as well as the performance of the *full* version, where all features are enabled. The results are shown in the upper part of Table 10.

As expected, the *enhanced* version performs basically well on both datasets. On top of it, *large database* improves a little bit on Macro F1 but also worsen a little bit on Micro F1. The changes are however very subtle. Compared to *enhanced* verson, *Wikipedia abstract* has negative impacts on both datasets, but performs better on AIDA with combination to *large database*. This shows that under certain conditions, *Wikipedia abstract* and *large database* can indeed improve performance. This may due to the rich context in AIDA. The analysis here is hard as many factors interact with each other. Both features bring more entities and information, which results in more entities being recognized, some are correct, some are wrong. However, they require more memory and processing time. In summary, when the resources are limited or the query tend to be short, the *enhanced* version is preferred.

4.3.3 Comparison to AmbiversNLU

Finally, the comparison to AmbiverseNLU is shown in the lower part of Table 10. The results on the two benchmarks are quite different. On Clueweb, our algorithm outperforms AmbiverseNLU by over 12 points of Macro F1 score and 4 points of Micro F1 score. As described earlier, Macro F1 score is preferable in Clueweb as its documents tend to have diverse characteristics. On the other hand, AmbiverseNLU performs better than our algorithm on AIDA, where texts are news articles with rich

configuration	**Clueweb** Micro F1 Macro F1	**AIDA** Micro F1 Macro F1
enhanced	**49.82** <u>46.21</u>	61.47 60.98
enhanced + large database	<u>49.61</u> **46.59**	61.39 61.26
enhanced + Wikipedia abstract	48.78 45.18	60.53 60.17
full	48.55 45.56	<u>62.31</u> <u>61.50</u>
AmbiverseNLU	44.75 33.58	**68.57** **67.78**

Table 10: NERD results of multiple features and comparison to AmbiverseNLU. Micro and Macro F1 scores are shown. *enhanced* denotes our base model plus *family name*, *demonym* and *NNP reduction*. *full* denotes our base model plus all five features enabled. We highlight the best result in bold and the second best with underline.

context. It is worth mentioning that named entities in AIDA tend to repeat more than one time as it is the characteristic of news. Therefore, the difference in scores can easily be enlarged because the same error will be calculated multiple times.

To further explore the reasons, we look at the statistics of true positive, false positive, and false negative of each model on both benchmarks in Table 11. On Clueweb, our algorithm is able to report more true positives but also produces much more false positives than AmbiverseNLU. This explains the reason for the difference in scores. AmbiverseNLU tends to be conservative on reporting named entities, hence may suffer more on short texts. On AIDA, our algorithm performs generally the same with AmbiverseNLU on true positives and false negatives, but has twice more false positives, thus results in lower F1 scores. The statistics coincides with our design to recognize as many named entities as possible.

4.4 Error Analysis

We conduct an error analysis by random sampling about 50 errors on both datasets on the *full* version of our algorithm. Errors from both datasets are then aggregated

model	Clueweb			AIDA		
	tp	**fp**	**fn**	**tp**	**fp**	**fn**
enhanced	40,176	43,721	37,213	16,340	9,316	11,167
full	40,146	47,833	37,243	17,208	10,522	10,299
AmbiverseNLU	26,083	16,119	48,299	17,136	5,319	10,393

Table 11: True positives, false positives, and false negatives of different models.

Type	Percentage
unrelated entities	29.13%
wrongly disambiguated entities	19.42%
unrecognized entities	17.48%
wrongly discarded entities	11.65%
doubtful ground truth	8.74%
answers not in ground truth	6.8%
others	6.8%

Table 12: Error analysis.

and categorized into 7 groups, shown in Table 12. The errors mainly come from unrelated entities (29%), wrongly disambiguated entities (19%), unrecognized entities (17%) and wrongly discarded entities (12%). Wrongly disambiguated entities is easy to understand, others are described in the following paragraphs. Note that two categories are not actual errors: 9% of the errors are from doubtful ground truth, some are wrong answers, and some are too short in context to identify whether the answer is correct; another 7% of the errors are due to that the algorithm output is indeed a correct named entity but not included in the ground truth. Note also that all the errors of doubtful ground truth come from Clueweb, which is not surprising as it is automatically annotated.

Unrelated entities are the main source of the errors. It is due to a non-entity word tagged as `NNP` and then disambiguated to a unrelated entities. An example is "From the day you buy your Saturn , we provide 24-Hour Roadside Assistance ...". In the sentence, "Roadside" is wrongly reported as `NNP` by spaCy and then disambiguated to a film, which is unrelated to the context. Capitalized nouns have chances to be tagged as `NNP` and suffered from this sort of error.

Unrecognized entities come from the limitation of the matching mechanism in our

entity index. In this category of error, the entities are correctly pointed out by spaCy but failed to find a matched entity in the entity index. One type is due to unknown synonyms. For example, "Playstation" can not be matched to "PlayStation"; "U.N." can not be matched to "UN". Another type is because of partial mention. For example, "Security Concil" can not be matched to "UN Security Concil"; "Korean" can not be matched to "South Korean".

Wrongly discarded entities are text spans that are indeed a named entity but with a tag of `NN` or `JJ` thus discarded in the final step of the NER algorithm. The errors come from wrong postags by spaCy. For example, "their **iOS** `JJ` app is no exception", or "**Wordpress** `JJ` do have default video embedding and image embedding".

5 Conclusion

5.1 Summary

In this thesis, we propose an algorithm with configurable features for named entity recognition and disambiguation with Wikidata in arbitrary English text. In recognition, we utilize the POS tags generated by spaCy as a filter to speed up the matching process with the pre-generated entity index. We expend synonyms with the Wikidata property "family name" and "demonym" to enhance recognition flexibility. In disambiguation, we consider both popularity and similarity of the candidate entities and leverage the information from Wikipedia abstracts. We further discard unlikely named entities to decrease false positives. In evaluation, we analyze the contribution of each component as well as the entire NERD system. Results show the effectiveness of synonym expansion and false positive reduction. In addition, our algorithm performs better and is capable of outputting more named entities on one dataset with shorter documents. On the other dataset consisting of news articles with rich context, our performance is also acceptable in comparison with the other sophisticated system.

5.2 Future Work

Here are some ideas and directions that can be further studied to improve this work.

- Although spaCy is a good POS-tagger, we do find many errors induced by wrong POS tags in the error analysis. Some of those sentences are however correctly tagged by Standford POS-Tagger. Since our algorithm relays on POS tags to do NER, it's worth evaluating other POS-taggers.

- More preparation on the database is a possible way to recognize more entities. For example, to integrate named entities from multiple knowledge bases, or to include the information from alias tables into current database. Also, simple nouns can be excluded from the database to avoid some false positives.

- How to make the best use of Wikipedia abstract or other resource to measure similarity between candidates and context is also a meaningful direction to improve. One possible approach is to represent each candidate entity by its word embeddings.
- The number of false positives needs to be further reduced to provide a more reliable system. This is a complicated issue. Though adopting a confidence threshold or force the coherence of named entities could help, how to avoid removing the true entities still deserves careful concerns, especially when the text is short and not much information can be used.

6 Acknowledgments

First and foremost, I would like to thank...

- Professor Dr. Hannah Bast for providing me the opportunity to work with her for my project and thesis, her valuable time to supervise me, and her insightful advice and ideas to expand my work.
- Matthias Hertel for his detailed feedback on my benchmark and evaluation process.
- Frank Dal-Ri for his immediate technical supports.
- my parents for their encouragement and trust throughout the course of my studies.
- my son for his coming to our family, his innocent smile and his sweet company.
- my husband to be my best teammate, my strong and reliable backing in all aspects, and his continuous companionship and support especially at the final time of the completion of the thesis.

Bibliography

[1] R. Mihalcea and A. Csomai, "Wikify!: Linking documents to encyclopedic knowledge," in *Proceedings of the Sixteenth ACM Conference on Conference on Information and Knowledge Management*, pp. 233–242, ACM, 2007.

[2] S. Cucerzan, "Large-scale named entity disambiguation based on wikipedia data.," in *EMNLP-CoNLL* (J. Eisner, ed.), pp. 708–716, ACL, 2007.

[3] D. Milne and I. H. Witten, "Learning to link with wikipedia," in *Proceedings of the 17th ACM conference on Information and knowledge management*, pp. 509–518, ACM, 2008.

[4] J. Hoffart, M. A. Yosef, I. Bordino, H. Fürstenau, M. Pinkal, M. Spaniol, B. Taneva, S. Thater, and G. Weikum, "Robust disambiguation of named entities in text," in *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, (Edinburgh, Scotland, UK.), pp. 782–792, Association for Computational Linguistics, July 2011.

[5] S. Kulkarni, A. Singh, G. Ramakrishnan, and S. Chakrabarti, "Collective annotation of wikipedia entities in web text.," in *KDD* (J. F. E. IV, F. Fogelman-Soulié, P. A. Flach, and M. J. Zaki, eds.), pp. 457–466, ACM, 2009.

[6] J. R. Finkel, T. Grenager, and C. D. Manning, "Incorporating non-local information into information extraction systems by gibbs sampling.," in *Proceedings of the 43nd Annual Meeting of the Association for Computational Linguistics (ACL 2005)* (T. A. for Computer Linguistics, ed.), pp. 363–370, 2005.

[7] J. Hoffart, F. M. Suchanek, K. Berberich, E. Lewis-Kelham, G. de Melo, and G. Weikum, "Yago2: Exploring and querying world knowledge in time, space, context, and many languages," in *Proceedings of the 20th International Conference Companion on World Wide Web*, WWW '11, (New York, NY, USA), pp. 229–232, ACM, 2011.

[8] F. Piccinno and P. Ferragina, "From tagme to wat: a new entity annotator.," in *ERD@SIGIR* (D. Carmel, M.-W. Chang, E. Gabrilovich, B.-J. P. Hsu, and K. Wang, eds.), pp. 55–62, ACM, 2014.

[9] P. Ferragina and U. Scaiella, "Tagme: on-the-fly annotation of short text fragments (by wikipedia entities)," *CoRR*, vol. abs/1006.3498, 2010.

[10] N. Kolitsas, O.-E. Ganea, and T. Hofmann, "End-to-end neural entity linking.," *CoRR*, vol. abs/1808.07699, 2018.

[11] O.-E. Ganea and T. Hofmann, "Deep joint entity disambiguation with local neural attention.," in *EMNLP* (M. Palmer, R. Hwa, and S. Riedel, eds.), pp. 2619–2629, Association for Computational Linguistics, 2017.

[12] M. P. Marcus, B. Santorini, and M. A. Marcinkiewicz, "Building a large annotated corpus of English: The Penn Treebank," *Computational Linguistics*, vol. 19, no. 2, pp. 313–330, 1993.

[13] M. Honnibal, I. Montani, S. Van Landeghem, and A. Boyd, "spaCy: Industrial-strength Natural Language Processing in Python," 2020.

[14] E. Gabrilovich, M. Ringgaard, and A. Subramanya, "Facc1: Freebase annotation of clueweb corpora, version 1 (release date 2013-06-26, format version 1, correction level 0)," 2013.

[15] D. Seyler, T. Dembelova, L. D. Corro, J. Hoffart, and G. Weikum, "A study of the importance of external knowledge in the named entity recognition task.," in *ACL (2)* (I. Gurevych and Y. Miyao, eds.), pp. 241–246, Association for Computational Linguistics, 2018.