Master Thesis

Albert-Ludwigs-Universität

Freiburg

Entity Disambiguation using Freebase and Wikipedia

Author:
Ragavan Natarajan

Supervisor:
Prof. Dr. Hannah Bast

This report is submitted in partiful fulfillment for the master thesis at the at the

Chair of Algorithms and Data Structures
Department of Computer Science

Declaration

This thesis is an account of research undertaken between September 2013 and March 2014 at the Department of Computer Science, Faculty of Engineering, Albert-Ludwigs-Universität, Freiburg, Germany.

Except where acknowledged in the customary manner, the material presented in this thesis is, to the best of my knowledge, original and has not been submitted in whole or part for a degree in any university.

Ragavan Natarajan

March, 2014

Acknowledgments

I would like to start by thanking my supervisor Prof. Dr. Hannah Bast for the course she offered on Information Retrieval, which helped me develop a lot of interest in this field, and which has also given me a great career in this field. I am greatly thankful to her availability on e-mail, whenever I had some queries. I would also like to thank Florian Bäurle for his valuable feedback and critical suggestions at crucial times, without which this work would not have been possible.

I am also greatly thankful to Prof. Dr. Christian Schindelhauer, who has offered some very good courses in the field of Distributed Computing, all of which I have taken part in and throughly enjoyed, during the first half of my master degree. It was in the Seminar course on Distributed Algorithms, jointly offered by him and Dr. Alexander Souza, I developed the necessary skills on where to look for, how to read and understand research papers, not to mention, the skills to write one.

Abstract

This thesis addresses the problem of entity disambiguation, which involves identifying important phrases in a given text and linking them to the appropriate entities they refer to. For this work, information extracted from both Freebase and Wikipedia served as the knowledge base. A fully functional entity disambiguation tool is made available online and the challenges involved in each stages of the development are addressed in this paper.

It explains how the phrases in a text are assigned importance measure, both before and after the disambiguation process, based on several studies available in the literature and experimental analysis conducted as a part of this work. Additionally it explains the implementation of the collective-entity-linking algorithm by Han et al.[1] for entity disambiguation.

Several highly reusable code have been developed, whose usability extend well beyond the work described here. Stages involved in the development, from parsing the database dump to storing them in a high-speed, memory efficient key-value store are addressed. By means of *D3js* a JavaScript based vector graphics library for bringing data to life, visualization of the ambiguity tree is provided, which helps the user easily comprehend the results.

Zusammenfassung

Diese Masterarbeit beschäftigt sich mit der Erkennung von wichtigen Begriffen in einem Text und der Verknüpfung dieser Begriffe mit den entsprechenden Entitäten. Dafür wurde eine Wissensdatenbank aus Freebase und Wikipedia hergestellt. Eine voll funktionierende Anwendung wurde entwickelt und jeder Schritt der Entwicklung der Anwendung wird hier erklärt.

Contents

Declaration i

Acknowledgments ii

Abstract iii

Zusammenfassung: Abstract in German iv

1 Introduction 1
- 1.1 Entity disambiguation: *defined* . . . 1
- 1.2 Organization of the thesis . . . 2

2 Knowledge-Base Creation 4
- 2.1 Creating knowledge base from Wikipedia . . . 5
  - 2.1.1 Obtaining the Wikipedia database dump . . . 5
  - 2.1.2 Types of pages in Wikipedia . . . 5
  - 2.1.3 Free Links in Wikipedia . . . 6
  - 2.1.4 Format of the data . . . 6
  - 2.1.5 Keyphrase extraction . . . 7
    - 2.1.5.1 Composition of a keyphrase . . . 7
    - 2.1.5.2 Extracting from articles . . . 8
    - 2.1.5.3 Extracting from page titles . . . 8
    - 2.1.5.4 Extracting from article titles . . . 9
    - 2.1.5.5 Extracting from redirect titles . . . 9
    - 2.1.5.6 Extraction from disambiguation titles . . . 10
    - 2.1.5.7 Nested disambiguations . . . 10
    - 2.1.5.8 Chained disambiguations . . . 11
  - 2.1.6 Additional extracted information . . . 11
- 2.2 Creating knowledge base from Freebase . . . 11
  - 2.2.1 Obtaining the Freebase database dump . . . 12
  - 2.2.2 Parsing the Freebase database dump . . . 12
- 2.3 Storing the knowledge base . . . 13
- 2.4 Challenges involved in Knowledge-base creation . . . 13

3 Entity Disambiguation 15
- 3.1 Anterior phrase importance measure . . . 15
  - 3.1.1 Keyphraseness . . . 16
  - 3.1.2 tf × idf based importance . . . 16

3.1.3 Phrase retention score . . . 17
3.2 Anterior phrase-entity compatibility measure . . . 17
3.3 Anterior entity-entity relationship measure . . . 18
3.4 Construction of the Referent Graph . . . 18
3.5 Evidence Propagation . . . 19
3.5.1 Evidence propagation through *Compatibility Edges* . . . 19
3.5.2 Evidence propagation through *Semantic Relatedness Edges* . . . 19
3.6 The Collective Entity Linking Algorithm . . . 19
3.6.1 Computing $r_d(e)$ . . . 20
3.7 Posterior phrase importance measure . . . 20

4 The Entity Disambiguation Tool 22
4.1 $\mathcal{R}$ecognize and $\mathcal{D}$isambiguate: $\mathcal{R}n\mathcal{D}$ . . . 22
4.2 The code base . . . 24
4.2.1 Licensing the code . . . 25

5 Results 26
5.1 Precision and Recall . . . 26
5.2 Sample document and result . . . 27
5.2.1 Result of entity recognition and disambiguation . . . 27
5.2.1.1 Analysis of the result . . . 27
5.2.2 Results summary . . . 28

6 Future Work 29
6.1 Anterior importance of phrases . . . 29
6.2 Using Freebase's knowledge-graph for EL decisions . . . 29

7 Conclusion 31

A Computing the *n*-grams 32

B $tf \times idf$ score 33
Term Frequency. . . . 33
Document Frequency. . . . 33
Inverse Document Frequency. . . . 33

Bibliography 34

Chapter 1

Introduction

Entity Disambiguation in a text is a multi step process. At a high level, it involves the identification of significant phrases in the input text and associating them to the entities they correspond to. The following section discusses this in more detail.

1.1 Entity disambiguation: *defined*

The job of an entity disambiguator, as mentioned earlier, encompasses the identification of significant phrases (and not just words) in a text, and linking them to appropriate entities. Following are some of the challenges involved in this.

1. Abbreviations of some importance should be identified in the text and be matched to appropriate entities.
2. Partial name mentions, such as *Obama* or *Congress* should be identified and matched appropriately.

For example, consider the following text:

> At **BKC** rally, **BJP**'s prime ministerial candidate **Narendra Modi** takes a dig at **Rahul Gandhi** for his remarks on corruption, slams **Congress** government in state for trying to shield its leaders by sweeping **Adarsh scam** case under the carpet.

In the text given above, the significant phrases, i.e., those that the disambiguator should be able to identify as appropriate for the text, are marked in bold. As shown in figure 1.1, however,

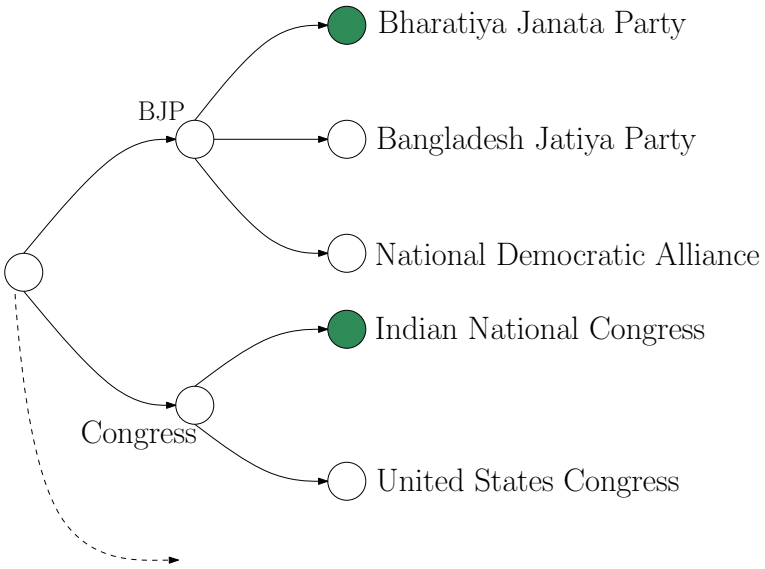

FIGURE 1.1: Identifying the correct one from ambiguous entities based on context.

each of the phrase could mean entirely different things. Therefore, the disambiguator should be able to understand the context and disambiguate name mentions.

In the context of the text, which is *Indian Politics*, the word *BJP* probably is more related to the entity **Bharatiya Janata Party**, a political party in India, than to Bangladesh Jatiya Party. Similarly, in the same context, the word *Congress* is more related to **Indian National Congress**, another Indian political party, than to United States Congress.

1.2 Organization of the thesis

While the notion of *significant phrases* could be clear to a human being, a machine needs a knowledge base, in order to be able to say what significant phrases are, and to be able to make well informed decisions. In other words, given a text like the one in the example, the machine should be able to automatically identify those very phrases that were marked in bold, as important ones. Therefore, a significant amount of effort has been made in the creation of the knowledge base. Data extracted from *Freebase*[1] and *Wikipedia* serves as the knowledge base for this work, and chapter 2 explains this process in great detail.

Chapter 3 explains how phrases are identified in the input text and an anterior importance score is assigned to them, based on information available from the knowledge base. Doing so, helps to eliminate insignificant phrases from being part of the phrases that are to be disambiguated, and hence, speeds up the entire process and gives more accuracy too.

[1]Section 2.2 explains what *Freebase* is

Additionally, it also discusses how the phrases are disambiguated using the *Collective Entity Linking* algorithm of Han et al.[1] and explains, how by means of posterior importance scores, insignificant phrases are further discarded from the results.

Chapter 4 talks about the application that has been developed as a part of this work, and shows how the results are presented to the user in an easily comprehensible manner, by means of *D3js*, a JavaScript vector graphics library.

The results are discussed in chapter 5, whereas chapter 6 discusses the room for improvement in the future. Finally, chapter 7 provides conclusion with a short note on the future work.

Chapter 2

Knowledge-Base Creation

A carefully-crafted knowledge-base is very important in helping the machine make entity-linking decisions. This chapter addresses the task of creating a knowledge-base, covering everything from obtaining the Freebase and Wikipedia database dump files, to parsing and extracting information from them, and also addresses the challenges involved in it, and ways to increase the size of the extracted vocabulary of significant phrases (hereinafter referred to by the word *keyphrases*).

The importance of a knowledge-base cannot be understacated, as the outcome of phrase recognition and entity disambiguation heavily depends upon the correctness of the information available in the knowledge base. Because Wikipedia was considered unsuitable for functioning as knowledge-base for *high-recall*[1] entity-linking tasks due to the sparse annotation of the keyphrases[1], special efforts have been made to extract keyphrases from unannotated entities too, in an attempt to overcome this drawback. Doing so has helped identify ca. 11 million keyphrases from the database dump.

Additionally, for entities that are in Freebase but not in Wikipedia, information is added to the knowledge base by processing the Freebase dump and extracting information from the textual description of those entities. Several restrictions are imposed upon what can constitute a *keyphrase*. Doing so, guarantees uniformity in the process of identifying keyphrases in an input document, as well as in extracting them from the database dumps, for knowledge-base creation.

Furthermore, storing the information in an efficient randomly accessible manner is of great importance. There are *ca.* 7.5 million unique articles (otherwise known as, entities) on Wikipedia, 45.5 million unique entities on Freebase, 11 million phrases, 35 million unique words which are

[1] *Recall* is a metric explained later.

not stop words with an average size of 9.65 characters per word, 5 million entities with incoming references and so on.

To further emphasize the need for efficient retrieval, for example, during the course of disambiguation, a *compatibility relation* between a *phrase* and an *entity* is computed which is based on the words appearing around the phrase and all the words appearing in the document, i.e., the Wikipedia article, corresponding to the entity. Additionally, the words are weighted based on their $tf \times idf$ scores. Therefore, the article corresponding to an entity and the precomputed *idf* of words need to be fetched efficiently to be able to compute the results quickly. Section 2.3 addresses how the knowledge base is stored for efficient random retrieval.

2.1 Creating knowledge base from Wikipedia

This section addresses how the knowledge base is created from the database dump of Wikipedia. Section 2.1.1 discusses how the Wikipedia database dump could be obtained and the sections that follow explain the different types of pages available in the dump and how a knowledge base is obtained from them.

2.1.1 Obtaining the Wikipedia database dump

The Wikipedia database dump file containing the latest pages and articles could be downloaded from http://download.wikimedia.org/enwiki/latest/enwiki-latest-pages-articles.xml.bz2. It contains only the current revisions and no talk or user pages − but it is sufficient. The size of the database dump as on February 2014 is approximately 9.85GB compressed, 44GB uncompressed.

The Perl module *MediaWiki::DumpFile::FastPages*, available in CPAN[2], has been used to parse the uncompressed database dump. It enables simultaneous iteration over the title and content of each page in the database dump in a sequential manner.

2.1.2 Types of pages in Wikipedia

Three different types of pages of interest could be identified based on the page-title of a page in the Wikipedia database dump. They are mentioned below.

1. **Disambiguation pages:**

[2] http://www.cpan.org/

The content of one such page on Wikipedia contains one ore more links to existent articles or other disambiguation pages. Such a page could be identified by the presence of the phrase *(disambiguation)* in its title text. For example, if the phrase *Austin (disambiguation)* is the title text of a page, then its content would be a list of one or more links as mentioned above.

2. **Redirect pages:**

   The content of one such page contains a single link that causes Wikipedia to redirect to a page (hereinafter referred to as the *target page*) referred by that link. It is possible that the target page in the dump file is also of type *redirect page*. Wikipedia uses the term *double redirects*[2] to denote such pages. In order to prevent infinite loops, Wikipedia stops the chain after the first redirect[2]. Hence, at the time of knowledge-base creation double redirects are ignored.

3. **Articles:**

   A page of this type is an article written about the topic contained in its title-text. This is what an user gets to see when they search for an existent article on Wikipedia. For example, following the URL https://en.wikipedia.org/wiki/Chennai, takes to a page on Wikipedia, which is of type *article*.

2.1.3 Free Links in Wikipedia

In Wikipedia, free links are used to produce internal links between pages[3], which enable users to access information related to the article they are reading. They are created by enclosing a page-title in double square brackets. Thus, enclosing the word *Chennai* in double square brackets, like this, [[*Chennai*]], will cause the text to be anchored with a hyperlink to the respective Wikipedia article, http://en.wikipedia.org/wiki/Chennai, in this case. Optionally a vertical bar '|' could be added to customize the anchor text. For example, typing [[*Chennai* | *the capital city of Tamilnadu*]] would result in the text *the capital city of Tamilnadu* being anchored to the respective Wikipedia article, which is, in this case, the aforementioned article.

2.1.4 Format of the data

The Wikipedia database dump is an XML file containing both metadata and data. As mentioned earlier, the Perl module *MediaWiki::DumpFile::FastPages*, available in CPAN, is used to process the database dump. Pages are written in **Wikitext language**[3], a lightweight markup language for writing Wikipedia pages.

[3] http://en.wikipedia.org/wiki/Wikitext

2.1.5 Keyphrase extraction

Keyphrase extraction is one of the major tasks involved in the knowledge-base creation. This section begins by defining certain rules on what constitutes a keyphrase. As mentioned before, a keyphrase can be composed of one or more words, but for reasons mentioned later in this section further restrictions are imposed on keyphrases.

2.1.5.1 Composition of a keyphrase

The following set of rules restrict what a keyphrase can be composed of.

1. A keyphrase can be composed of only alphanumeric characters. However, it is not just limited to alphanumeric characters in the ASCII representation.
2. Every non-alphanumeric character found in the keyphrase should be replaced with a single whitespace character.
3. It cannot contain punctuation of any form.
4. All the letters should be case-folded to lowercase.
5. In case of the phrase containing multiple words, the words are to be separated by a single whitespace character.
6. The number of words in a keyphrase is limited to a maximum of 10.

These restrictions on the keyphrase offer the following benefits:

1. The set of target entities of two keyphrases, which are essentially the same but are under different case foldings could then be combined. Consider two keyphrases *austin* and *Austin* each having their own set of referred entities. Let the referred entities of the keyphrase *austin* be *Austin Island* and *Austin College*, and let that of *Austin* be *Austin College* and *Austin Motor Company*.

   Applying the rules, the sets of entities of the two keyphrases could then be merged with the case-folded *austin* acting as the keyphrase with referred entities being *Austin Island*, *Austin College* and *Austin Motor Company*.

2. Eliminating the non-alphanumeric characters in the keyphrases provides an uniform way of recognizing the keyphrases in an input document at the time of wikification. Consider, for example, the following excerpt in an input document.

... Alaska's residents grapple with changing climate ...

The rules for keyphrase extraction when applied to the input document case-folds it to lowercase and replaces each of the non-alphanumeric characters in it with a single whitespace, causing the above excerpt to become as follows.

... alaska s residents grapple with changing climate ...

This makes it possible to recognize the word *alaska* in the document if such a keyphrase existed in the keyphrase vocabulary. Imagine, without these rules, the entire word would be *Alaska's* which may not contain a matching keyphrase in the keyphrase vocabulary of the knowledge-base.

2.1.5.2 Extracting from articles

An *article* is a special type of Wikipedia page as mentioned earlier. Keyphrases are extracted from the free-links of articles, which is described in section 2.1.3.

- For free links without a vertical bar, i.e., without '|', the enclosed text would act both as the keyphrase and the *target entity*[4]. For example, for the free link [[*Chennai*]], the extracted keyphrase and target entity would be the following.

  chennai ⟶ *Chennai*

- For free links with a vertical bar, the text before the bar, i.e. the entity text, acts as the target entity whereas the text that follows it acts as the keyphrase. In addition to that, the entity text is also used as a keyphrase, just like it was used in a free link without vertical bar. For example, for the free link [[*Chennai*|*the capital city of Tamilnadu*]], the following *keyphrase - target entity* pairs are extracted.

  the capital city of tamilnadu ⟶ *Chennai*
  chennai ⟶ *Chennai*

  Doing so, helps improve the keyphrase vocabulary, which in turn could increase the possibility of important phrases in a document being recognized at the time of wikification.

2.1.5.3 Extracting from page titles

Limiting the keyphrase extraction to free links in articles would severely limit the keyphrase vocabulary leaving several non-linked article titles orphan. In other words, if there was an article

[4] A *target entity* is one of the several articles on Wikipedia that a keyphrase could possibly link to.

on Wikipedia, not linked from any other article, then it will be left unlinked if the keyphrase extraction procedure is limited to extracting keyphrases from the body of the articles, as done above.

2.1.5.4 Extracting from article titles

As the parser iterates the article titles, keyphrases are extracted from the titles and the titles are added to the set of target entities of those keyphrases. Any information enclosed in parenthesis is discarded. For example, if a page title is *Casablanca (film)*, then the following *keyphrase – target entity* pair is extracted from it.

casablanca ⟶ *Casablanca (film)*

If another page title, such as *Casablanca (volcano)*, is encountered, then another *keyphrase – target entity* pair is generated with the same keyphrase, i.e. *casablanca* causing it to have two different target entities.

casablanca ⟶ *Casablanca (film)*, *Casablanca (volcano)*

It is necessary to understand why the text within parentheses is discarded. If it were not, then the generated keyphrase would, for example, be *casablanca film*. It is very unlikely that an input document has this exact word sequence. By discarding the text within parentheses, the chances of a phrase in an input document being found in the keyphrase vocabulary increases. Since there are algorithms to disambiguate entities in the later stages of wikification, this does not cause any problem. Even if the word sequence *casablanca film* appeared in an input document, the wikifier would still be able to identify the phrase *casablanca* in it.

2.1.5.5 Extracting from redirect titles

The keyphrase obtained from the title of a **redirect page** could not use the title as a target entity, since the title redirects to another page on Wikipedia. Therefore, the keyphrase is assigned to the title in the body of the redirect page (hereinafter referred to as the *redirect target*). Additionally, a keyphrase is extracted from the redirect target and assigned to it. For example, the title *Air Transport* on Wikipedia redirects to the title *Aviation*. In this case, the following *keyphrase – target entity* pairs are extracted.

air transport ⟶ *Aviation*
aviation ⟶ *Aviation*

However, in order to avoid chain-redirects, if a *redirect target* redirects to another page, no *keyphrase – target entity* pairs are obtained.

2.1.5.6 Extraction from disambiguation titles

A keyphrase obtained from the title of a disambiguation page has, as its target, each of the articles available in the page. For example, the title *Casablanca (disambiguation)* is that of a *disambiguation page* on Wikipedia, containing the following entities.

[[*Casablanca (volcano)*]]
[[*Casablanca Records*]]
[[*Casablanca (film)*]]

As done before, the keyphrase *casablanca* is extracted from the disambiguation title, and each of the entities in the disambiguation page is assigned to be target of this keyphrase, as shown below.

casablanca ⟶ *Casablanca (volcano)*, *Casablanca Records*, *Casablanca (film)*

2.1.5.7 Nested disambiguations

It is possible for an entry in a Wikipedia disambiguation page to link to another disambiguation page. For example, the page *Austin (disambiguation)* on Wikipedia, contains the following entry in its content, which is the title of another disambiguation page.

[[*Austin Station (disambiguation)*]]

A disambiguation entity cannot be assigned as the target of a keyphrase. Therefore, the page of the nested disambiguation entity is fetched and all the non-disambiguation entities in its content are assigned to this keyphrase. For example, if the page *Austin Station (disambiguation)* contained the following content,

[[*Austin (Amtrak station)*]]
[[*Austin (CTA Blue Line)*]]
[[*Austin (CTA Green Line)*]]

all of these entities are assigned to be the target of the keyphrase *austin*, as shown below.

austin ⟶ *Austin (Amtrak station)*, *Austin (CTA Blue Line)*, *Austin (CTA Green Line)*

In addition to that, obviously, as done before, the keyphrase keyphrase *austin station* also gets all these target entities.

austin station ⟶ *Austin (Amtrak station)*, *Austin (CTA Blue Line)*, *Austin (CTA Green Line)*

2.1.5.8 Chained disambiguations

It is possible for a nested disambiguation page to further contain links to disambiguation pages. Such entries are ignored to avoid having to go down a possibly infinite chain.

2.1.6 Additional extracted information

In addition to extracting the keyphrases, several other information are extracted from the database dump. A list of all articles titles and their contents are extracted. Similar to what was done during the keyphrase extraction, all non-alphanumeric characters are removed from the article contents at the time of extraction. The content of an article is used to compute its similarity with an input document, as discussed in the later chapters.

Moreover, for each article, a set of **incoming links** is also maintained. The incoming links of an article is the set of all articles in Wikipedia that have free-links to this article. This information is used to compute the semantic relatedness between two articles on Wikipedia.

2.2 Creating knowledge base from Freebase

Freebase is a large collaborative knowledge base consisting of metadata composed mainly by its community members. It is an online collection of structured data harvested from many sources, including individual 'wiki' contributions[4].

For instance, the entity Nescafé is identified by Freebase as a type belonging to /business/brand whose owner is identified as Nestlé. This information could help a machine make decisions based on the facts available to it by means of knowledge-graph it could easily understand. In their entity disambiguation framework, Zheng et al.[5] attempt to leverage two features of Freebase, namely the naturally disambiguated entities and the rich taxonomy, to perform entity disambiguation in an iterative manner.

However, their approach has some shortcomings, mainly due to the lack of maturity of Freebase as a knowledge base. Considering the example of *brands*, while Freebase has in its knowledge graph, information about a lot of international brands, it is still in expansion and does not have information for many popular regional brands. This shortcoming of Freebase also applies to any kind of entity, which is not a brand. For example, Mahindra XUV500, an automobile model, while very popular in India[6], does not have its owner identified as Mahindra & Mahindra, even though these entities independently exist on Freebase.

Therefore, in the experiments conducted with the approach of Zheng et al., attempt to achieve good results for every piece of input text was not successful due to these shortcomings of

Freebase. In other words, for text containing those entities that do not have a proper knowledge graph, their method performed poorly.

However, since, Freebase has significantly more entities than Wikipedia[5], information about these additional entities could be extracted from it. Therefore, the parser that was originally developed as a part of this work to extract the knowledge-graph from Freebase for implementing the algorithm of Zheng et al., was later modified to extract only the textual description from Freebase for entities that did not exist in Wikipedia. The following sections explain this process in great detail.

2.2.1 Obtaining the Freebase database dump

The Freebase database dump file could be obtained from here[5]. It constitutes a snapshot of the data stored in Freebase and the Schema that structures it, and are provided under the Creative Commons' CC-BY license.

As of Feb 2013, it contains a total of 1.9 billion triples in N-Triples RDF format, encoded as UTF-8 and compressed with GZip, whose compressed size is 22 GB and uncompressed size is 247 GB.

The following sections explain how the database dump file from Freebase is parsed to extract information needed for the creation of the knowledge-base.

2.2.2 Parsing the Freebase database dump

openRDF[6] provides a Java implementation for parsing RDF data. In this work, `TurtleParser`, a concrete-implementation of the `RDFParser` interface, has been used for parsing the Freebase database dump. The parser works based on Observer Pattern, a design principle in Software Engineering. Through the `setRDFHandler` method of the interface `RDFParser`, a handler of type `RDFHandler` is set and the `RDFParser`'s `parse` method is called, by means of which the Freebase database dump is parsed.

As a part of the work, a parser was originally developed to extract the complete knowledge-graph in Freebase. However, after the shortcomings of the disambiguation approach based on Freebase's knowledge graph, the parser was modified to parse and extract only information about additional entities, as described earlier. The code took approximately 10 hours on a personal computer with moderate configuration, to extract information about 45.5 million unique entities from the Freebase database dump.

[5] https://developers.google.com/freebase/data
[6] http://www.openrdf.org/

2.3 Storing the knowledge base

The importance of a highly efficient storage cannot be understated. The scale of data involved has already been discussed in the beginning of this chapter. The stages in disambiguation demands that the data be retrieved in a highly efficient manner. For example, as a first step, n-grams are generated from the input text and matched against the set of keyphrases, which is about a 11 million. Later, textual content of entities needs to be fetched at runtime and be matched against the local context of phrases. Due to the amount of data involved, a highly efficient storing scheme was required. Also, the scale of the data prevents it from being stored in the memory, as the available memory in most of the general purpose personal computers rarely exceed a couple of Gigabytes, as of this writing.

Therefore, after evaluating several SQL and NoSQL databases (including several DBM and QDBM based databases such as BerkeleyDB) based on experiments with very large sets of data, it was decided to use TokyoCabinet, a modern implementation of DBM written in C++. Essentially it is an efficient implementation of B+ trees for storing data on disk, in other words, a disk-based key-value store, with exceptional read performance and moderate to average write performance.

Since the knowledge base is created only once and never written to again, it was the read performance that was of huge importance. Therefore, TokyoCabinet was chosen to be the key-value store that would store all the information.

2.4 Challenges involved in Knowledge-base creation

There are several challenges involved in the creation of the knowledge base. For example, section 3.1.2 discusses about an importance measure, for which the document frequency[7] of each of the 11 million phrases needs to be computed.

There are, as mentioned earlier, over 7 million articles, and in computing the document frequency of phrases, the occurrence of each of the 11 million phrase in each of the 7 million articles need to be computed. A naïve approach of matching the phrases one after the other in all the articles would take months, if not, years to complete the task, with the power of general purpose personal computers available at the time of this writing. A more correct approach would be to generate, for each of the 7 million articles, all the n-grams[8] of up to 10 words and match them against the dictionary of phrases. By means of highly efficient multi-threaded code written for this work, this task was achieved in less than 5 hours.

[7] See Appendix B
[8] See Appendix A

Similarly, several other challenges were faced in creating the knowledge-base, all of which is not explained here for brevity's sake. Section 4.2 briefly discusses about the code base. The code is very well documented and organized in the form of packages. Having a look at the code base would help understand the challenges involved in creating the knowledge base and the efforts made, in addition to giving the curious reader the answers to all their questions.

Chapter 3

Entity Disambiguation

As discussed before, entity disambiguation in an input text involves multiple stages. At a very high level, significant phrases are to be identified in the input text and be associated to all the possible entities they refer to, and then the right entity is chosen for each of the phrase, by an algorithm, based on the context of occurrence of the phrases in the text. The algorithm chosen for implementation is the one due to Han et al.[1]. Additionally, several experiments were made at different stages to make the implementation very effective in terms of the quality of the output that it generates. This chapter addresses all of it, in great detail.

3.1 Anterior phrase importance measure

An anterior importance score would help prevent phrases of less significance from being part of the input to the disambiguation algorithm. As a first step, n-grams of up to 10 words are generated from the input text. There are $\sum_{i=1}^{10} M - i + 1$ n-grams of size 1 to 10, for a text with M words. For example, if the input text had 50 words, there are a total of 455 n-grams of size 1 to 10. After ignoring about 1500 stop-words, the rest of the n-grams are matched against a dictionary of phrases and the ones that match are retained.

However, many of the n-grams overlap with one another. For example, the phrase **Indian Cricket team** could have two overlapping n-grams, such as, **Indian Cricket team** and **Cricket**. If the overlapping n-grams are of different lengths, n-grams with the most number of words, in this case, Indian Cricket team is chosen over Cricket. However, there could be many n-grams that overlap yet being equal in size, in terms of the number of words. An anterior importance score helps choose the more significant n-gram in such cases.

Apart from overlapping n-grams, there could be many other insignificant n-grams still present, which would form the input to the disambiguator. However, to achieve good results with the

collective entity linking algorithm, it is important to eliminate as many insignificant phrases as possible before the disambiguation step, so that the entity linking decisions could happen in a collective sense. Having insignificant phrases would greatly affect this process, as many irrelevant entities would take partake in the decision making, producing incorrect results. The following sections discuss the different methods of ranking phrases before the disambiguation step.

3.1.1 Keyphraseness

Mihalcea and Csomai[7] used this measure for assigning importance to phrases. The keyphraseness of a phrase p, measures the significance of a phrase in any document. It is defined as follows.

$$\text{keyphraseness}(p) = \frac{|p_{\text{link}}|}{|\text{DF}(p)|}$$

Where, $|p_{\text{link}}|$ is the number of articles in the knowledge base in which the phrase p appears as a hyperlink, and $\text{DF}(p)$ is the document frequency of p. It holds that $|p_{\text{link}}| \leq \text{DF}(p)$, and hence,

$$0 \leq \text{keyphraseness}(p) \leq 1$$

A phrase with keyphraseness of 1.0 would mean that it is linked wherever it appears, and hence, must be an important phrase, whereas a phrase with lower keyphraseness score would mean that it is seldom linked, and hence, is a phrase of less significance. If $\mathbb{P}$ is the set of phrases, for each phrase $p \in \mathbb{P}$, the normalized keyphraseness $\mathcal{N}_k(p)$, is computed as follows.

$$\mathcal{N}_k(p) = \frac{\text{keyphraseness}(p)}{\sum_{p \in \mathbb{P}} \text{keyphraseness}(p)}$$

Therefore, one appropriate way would be to sort the phrases in the input document in descending order of their respective normalized keyphraseness scores. For overlapping n-grams with same number of words, the normalized keyphraseness feature is used to pick the most relevant of them. The other ones are discarded from being part of the disambiguation process.

3.1.2 tf $\times$ idf based importance

The $tf \times idf$ measure, explained in more detail in Appendix B, computes how important is a word for a given document in a collection. For the purposes of this algorithm, it as been modified to assign importance to phrases (one or more words) rather than to single word. However, unlike

the keyphraseness measure explained in the previous section, this measure takes into account the input document to which the phrase belongs, in order to compute the term frequency. If $\mathbb{P}$ is the set of phrases, then the importance $\mathcal{I}$ based on $tf \times idf$ is computed as follows.

$$\mathcal{I}(p) = \frac{\text{tf} \times \text{idf}(p)}{\sum_{p \in D} \text{tf} \times \text{idf}(p)}$$

Where, D is the input text which is to be disambiguated. For this, a separate database containing the *idf* of phrases is maintained in the knowledge base. It contains the list of all phrases from the dictionary of phrases, and their *idf* score.

3.1.3 Phrase retention score

Let $\mathbb{P}$ be the collection of phrases whose retention score needs to be computed and let $p \in \mathbb{P}$ be a phrase. Then, the phrase retention score $\mathcal{R}$ of a phrase p is computed based on $\mathcal{N}_k(p)$ and $\mathcal{I}(p)$ as follows.

$$\mathcal{R}(p) = \frac{\mathcal{I}(p) \times \mathcal{N}_k(p)}{\sum_{p \in \mathbb{P}} \mathcal{I}(p) \times \mathcal{N}_k(p)}$$

By means of experimental analysis, phrases with $\mathcal{R}(p) < 0.1$ are prevented from being part of the disambiguation process. Additionally, out of the phrases thus retained, only a maximum of $x\%$ of the phrases of the number of words in the input document are considered for disambiguation, where $10 \leq x \leq 100$, which could be chosen by the user. It is important to note, however, that since most of the phrases are already discarded by means of the retention score, having a value of x to be 100%, doesn't mean that all the phrases in the input document will be considered for disambiguation.

3.2 Anterior phrase-entity compatibility measure

An anterior phrase-entity compatibility helps to limit the amount of possible entities for a given phrase and helps the collective entity linking algorithm make more well informed decisions by restricting the set to a limited number of entities. The compatibility between a phrase p and an entity e, denoted by $\text{CP}(p, e)$ is defined as follows[1].

$$\text{CP}(p, e) = \frac{\vec{m} \cdot \vec{e}}{|\vec{m}||\vec{e}|}$$

Where, $\vec{m}$ is a vector containing the $tf \times idf$ scores of the words in the *local context* of the keyphrase, $\vec{e}$ is a vector of $tf \times idf$ scores of the words in the entity, and $|\vec{m}|$ and $|\vec{e}|$ denote the normalization of the vectors $\vec{m}$ and $\vec{e}$, respectively.

The *local context* of a phrase p in a given text is the list of words surrounding the phrase for a window size of 50 words, as determined by Pedersen et al.[8]. The entities are sorted in decreasing order of their compatibility scores with the phrase, and only the top 10 entities are taken into consideration for disambiguation.

3.3 Anterior entity-entity relationship measure

This relationship creates weighted edges between every entity in the set of all entities of all phrases that are *semantically related* to each other. The semantic relatedness relationship, due to Milne and Witten[9], between two entities a and b, denoted by $\mathrm{SR}(a, b)$, is defined as follows.

$$\mathrm{SR}(a,b) = 1 - \frac{\log(\max(|A|,|B|)) - \log(|A \cap B|)}{\log(U) - \log(\min(|A|,|B|))}$$

Where, A and B are sets of all documents where the entities a and b appear as link, respectively, and U is the set of all documents in the universe. If $A \cap B = \emptyset$, then $\mathrm{SR}(a, b)$ is set to 0, meaning that the two entities are not semantically related to each other. Note that, $\mathrm{SR}(a, b) = \mathrm{SR}(b, a)$

3.4 Construction of the Referent Graph

The *Collective Entity Linking* algorithm makes entity linking decisions by means of a Referent Graph $G = (V, E)$, a directed graph, with the following properties.

1. If $\mathbb{P}$ is the set of all phrases that were retained, and $\mathbb{E}$ is the set of all entities of all the phrases in $\mathbb{P}$, then the set of vertices V is simply $\mathbb{P} \cup \mathbb{E}$.
2. If there is a compatibility relationship between a phrase $p \in \mathbb{P}$ and an entity $e \in \mathbb{E}$, then there is an edge $(p, e) \in E$, called the *compatibility edge*, whose weight is $\mathrm{CP}(p, e)$.
3. If $\{e_i, e_j\} \subseteq \mathbb{E}$ are semantically related, i.e., $\mathrm{SR}(e_i, e_j) \neq 0$, then there are *semantic relatedness edges* $\{(e_i, e_j), (e_j, e_i)\} \subseteq E$, whose weights are $\mathrm{SR}(e_i, e_j)$.
4. $\forall\ e \in \mathbb{E}, p \in \mathbb{P},\ (e, p) \notin E$. In other words, no edges are permitted in the graph that originate at an entity node and end at a phrase node.

The following section explains, how this graph is used for propagating the evidence.

3.5 Evidence Propagation

The $tf \times idf$ based importance measure $\mathcal{I}$ discussed in section 3.1.2 is reinforced by means of propagation through the edges in the dependency graph.

3.5.1 Evidence propagation through *Compatibility Edges*

$\forall\ p \in \mathbb{P}, e \in \mathbb{E}$, if there is a *compatibility edge* $(p, e) \in E$, then the *evidence propagation ratio* $\mathcal{P}$ is defined as follows.

$$\mathcal{P}(p \to e) = \frac{\mathrm{CP}(p, e)}{\sum_{e \in N_p} \mathrm{CP}(p, e)}$$

Where, N_p is the set of neighboring entities of the phrase p. Note that, there cannot be an evidence propagation ratio $\mathcal{P}(e \to p)$, as the referent graph cannot have edges from an entity to phrase.

3.5.2 Evidence propagation through *Semantic Relatedness Edges*

$\forall\ \{e_i, e_j\} \subseteq \mathbb{E}$, if there is a *semantic relatedness edge* $(e_i, e_j) \in E$, then the *evidence propagation ratio* $\mathcal{P}$ is defined as follows.

$$\mathcal{P}(e_i \to e_j) = \frac{\mathrm{SR}(e_i, e_j)}{\sum_{e \in N_{e_i}} \mathrm{SR}(e_i, e)}$$

Where, N_{e_i} is the set of neighboring entities of the entity e_i. Note that, $\mathcal{P}(e_i \to e_j)$ is not a commutative function.

3.6 The Collective Entity Linking Algorithm

The collective entity linking algorithm, as its name indicates, aims to exploit the global interdependence between different entity linking decisions and the local mention to entity compatibility, which is modeled in the referent graph discussed earlier.

Let $\mathbb{P}$ be the set of phrases and let $\mathbb{E}$ be the set of entities. Let $\mathbb{E}_p$ be the set of target entities of a phrase $p \in \mathbb{P}$. Then the most relevant target entity $\mathcal{T}(p)$, of the phrase p, is identified as follows.

$$\mathcal{T}(p) = \underset{e \in \mathbb{E}_p}{\operatorname{argmax}} \, \mathrm{CP}(p, e) \times r_d(e)$$

Where, $\mathrm{CP}(p, e)$ is the compatibility score discussed in section 3.2 and $r_d(e)$ is the evidence score for the entity e to be a referent entity of the document d. The following section discusses how $r_d(e)$ is jointly computed for all the candidate referent entities of a document d.

3.6.1 Computing $r_d(e)$

For every $v \in V$ of the referent graph $G = (V, E)$, indices are assigned randomly to each of the vertex from $1, \ldots, |V|$ and the adjacency matrix A of size $|V| \times |V|$ is written, such that $\forall\{i, j\} \subseteq \{1, \ldots, V\}$, $A_{i,j}$ is the edge weight between node i and j, if $e(i, j) \in E$, or 0, otherwise.

Additionally,

1. let s be the initial evidence vector, a $|V| \times 1$ vector, where, $s_i = \mathcal{I}(i)$ if $i \in \mathbb{P}$
2. let r be the evidence vector of size $|V| \times 1$, where r_i is the evidence score of the node i to be a target entity in document d if $i \in \mathbb{E}$ or $r_i = \mathcal{I}(i)$, if $i \in \mathbb{P}$.
3. let M be the evidence propagation matrix, a $|V| \times |V|$ matrix, where $M_{i,j}$ is the evidence propagation ratio from node j to node i, described in section 3.5.

Then, the evidence vector r is computed as follows, according to [1], [10].

$$r = \lambda(\mathrm{I} - cM)^{-1} \times s$$

Where, $\lambda = 0.1$ [1] is the fraction of the reallocation evidence and $c = 1 - \lambda$ and I is the identity matrix. This way, the algorithm combines the evidences from the interdependence between entity linking decisions, and the local compatibility between phrase and entities, and the relative importance of phrases.

3.7 Posterior phrase importance measure

After the collective entity linking algorithm is run, the phrases are assigned a posterior importance score, to further gain confidence. The posterior importance score of a phrase p, $\mathcal{I}_{\mathrm{post}}(p)$ is defined as follows.

$$\mathcal{I}_{\text{post}} = \mathcal{I}(p) \times r_d(\mathcal{T}(p))$$

Where, $\mathcal{I}(p)$ and $\mathcal{T}(p)$ are defined in sections 3.1.2 and 3.6, respectively.

Chapter 4

The Entity Disambiguation Tool

This chapter explains the functionality provided by the application and the supporting APIs. It briefly talks about the user experience and the insight the application is able to provide to the user by means of *D3js*, a JavaScript vector graphics library. It also briefly discusses the other reusable components developed as a part of this application, and how the entire application is bundled in a single deployment tool, to make life easier for anyone wanting to install and run the application.

4.1 $\mathcal{R}$ecognize and $\mathcal{D}$isambiguate: $\mathcal{R}n\mathcal{D}$

The tool is named so in accordance with the task it performs. It is a Java based web-application served by the Apache Tomcat web container running a Java servlet. On the client side, the application interacts with the server by means of *Ajax* using the jQuery library. The client side user interface components are built using the jQuery UI, another rich JavaScript based library, for building powerful UI components.

Additionally, for visualization, D3js, a rich JavaScript based vector graphics library is used. Using this library, a graphical visualization of the ambiguity tree is provided, which shows the list of all the phrases that were identified, sorted in decreasing order of their confidence scores, and when the user clicks on a node in the tree, it would expand to show all the entities that were referred to by the phrase and marks the entity that was declared by the algorithm to be the winner.

Moreover, at the output, the phrases are color coded for different confidence levels. The user could also choose a different anterior importance metric for the phrases and see how the results are affected, in addition to being able to specify, in terms of percentage, the number of phrases to be recognized at the output.

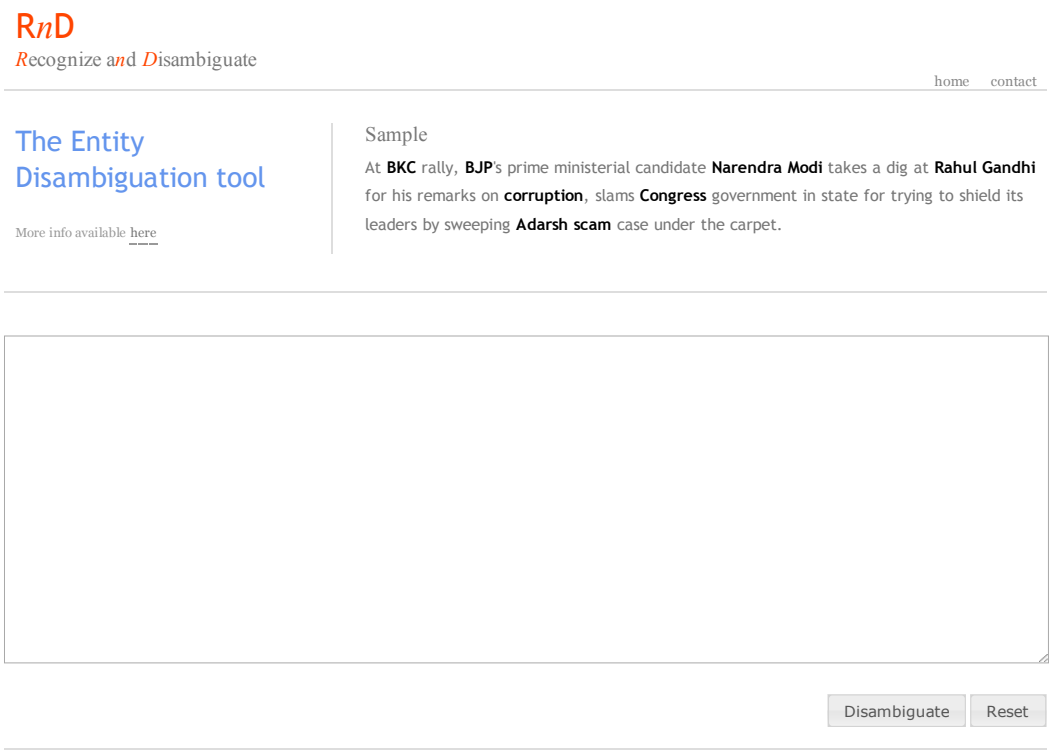

FIGURE 4.1: The input area

Figure 4.1 shows the application as it appears when the user opens it. The user enters the text in the *textarea* and clicks on the *Disambiguate* button, after which, by means of Ajax, communication with the server takes place and the response is printed in the response area, as shown in figure 4.2. The colored text represent the phrases and the confidence achieved for the phrase in its entity linking decision. The colors for different confidence percentage are explained by means of a legend. When the user clicks on any of the disambiguated phrases, hyperlinks are provided to the corresponding entity on both Freebase and Wikipedia, if they exist on both.

Additionally, when the user clicks on the *Ambiguity tree* tab which is seen in figure 4.2, the user is shown an interactive tree, as seen in figure 4.3, which shows the set of disambiguated phrases in sorted order of their respective confidence percentages, and when the user clicks on a phrase, it expands to show the list of all contending entities it had to disambiguate, and the most relevant target entity it declared as the winner, by means of a dark circle around the node.

It is clearly evident from 4.3 that the tool is capable of recognizing phrases and matching them to the most appropriate entities based on context, rather than to what a human would do. For example, in the context of the article that was given as the example, which is *cricket*, it is more appropriate for the the word *England* to point to *England National Cricket Team* than to the country *England*, which is what this tool exactly achieved.

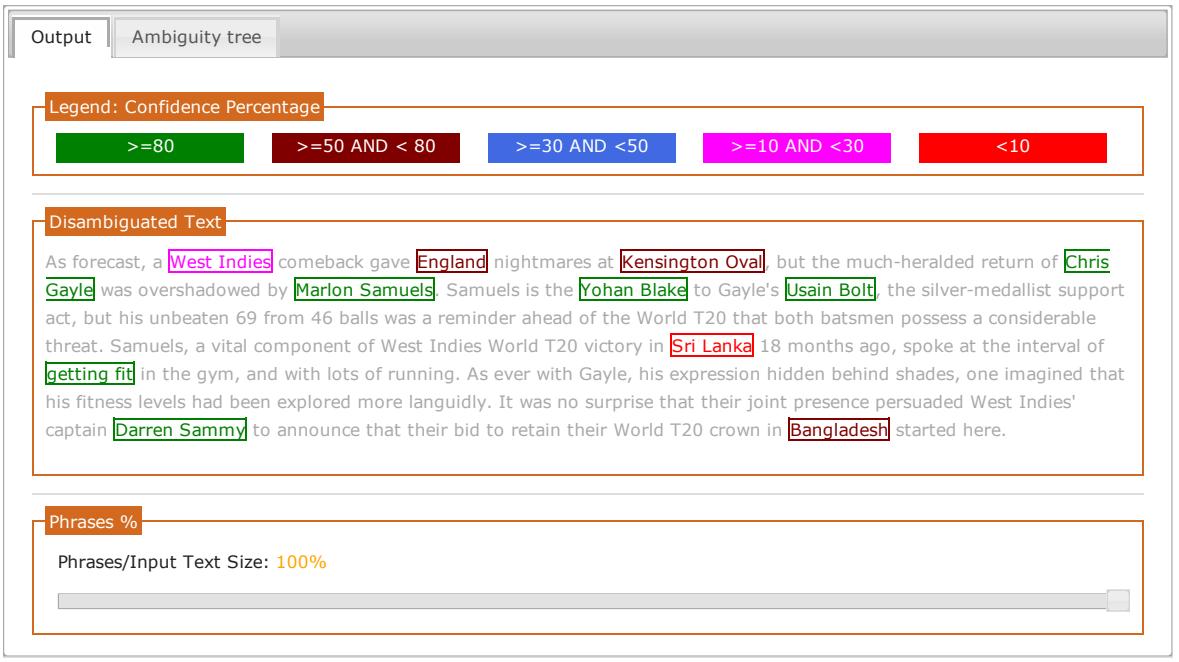

FIGURE 4.2: The Disambiguated text

However, for entity linking decisions, where the tool is not confident enough (i.e., where the confidence falls below 10%), the tool is forcefully made to select the full-text match, even if it would be less accurate. For example, as seen in figure 4.3 the phrase *Sri Lanka* is forcefully made to point to the country *Sri Lanka* rather than to the *Sri Lankan national cricket team*. This is done to have more accurate results. From the text it is clearly evident to a human reader, based on the context, that the country *Sri Lanka* is referred to by the phrase. But, since the article is overall about cricket, the disambiguator is not sure whether the phrase *Sri Lanka* refers to the country or its cricket team. Therefore, when it cannot make confident decisions, it is forcefully asked to do a text match, which, in this case, selects *Sri Lanka* as the target entity, which is also the correct result here.

4.2 The code base

The code base has over 10000 lines of code written in Java that span across 73 source files forming the core components of the system. Additionally, over 1000 lines of code form the supporting structure, which includes the HTML and JavaScript code for the web application, the shell scripts for deployment, and any other supporting code.

The entire code base is available in an online `git` repository, www.bitbucket.org, and could be obtained using the following command, after having `git` installed.

```
git clone https://rnatarajan@bitbucket.org/rnatarajan/freebifier.git
```

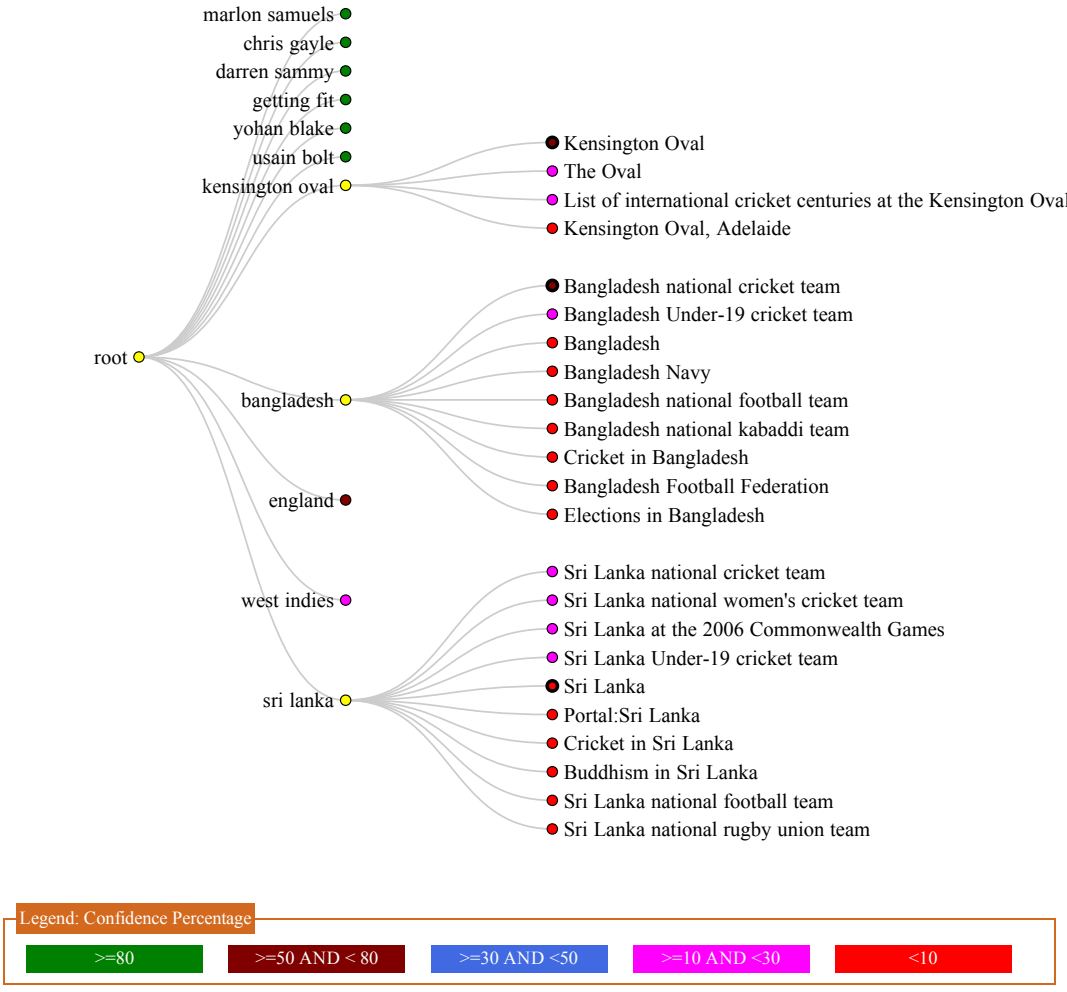

FIGURE 4.3: The Ambiguity Tree created using the D3js library

4.2.1 Licensing the code

The code contains intellectual property, and is, therefore, as of this writing, strictly not available for commercial use. It is, however, granted license forever **for academic use only**, to the Department of Computer Science[1] of Albert-Ludwigs-Universität, Freiburg[2].

[1] http://www.informatik.uni-freiburg.de/
[2] http://www.uni-freiburg.de/

Chapter 5

Results

This chapter begins by introducing the metrics for evaluating the result of entity disambiguation. For the purpose of discussion of the results, a sample text is disambiguated, and later, a generalization of the result for different categories of input documents, of different sizes each, is provided.

5.1 Precision and Recall

Precision and **Recall** are widespread measures for evaluating the quality of the result. Let $\mathbb{P}_r$ be the set of phrases linked correctly, $\mathbb{P}_i$ be the set of phrases that are insignificant but included in the result, or phrases that are linked incorrectly, and let $\mathbb{P}_u$ be the set of significant phrases that were unidentified. Then the precision and recall are defined as follows.

$$\text{Precision, } \mathfrak{P} = \frac{|\mathbb{P}_r|}{|\mathbb{P}_r| + |\mathbb{P}_i|}$$

and,

$$\text{Recall, } \mathfrak{R} = \frac{|\mathbb{P}_r|}{|\mathbb{P}_r| + |\mathbb{P}_u|}$$

Both precision and recall are measured in percentage. The quality of an entity disambiguator could be considered perfect if both $\mathfrak{P}$ and $\mathfrak{R}$ are 100%. In practice, however, attempt to increase either of the two may adversely affect the other. For example, $\mathfrak{R}$ could be increased by attempting to recognize many number of phrases in the input document. This may increase the possibility of all relevant and significant phrases in the document being identified and linked, thereby increasing $\mathfrak{R}$, but it may also, as a byproduct, increase the amount of irrelevant phrases being identified and linked, causing $\mathfrak{P}$ to plummet. Therefore, the aim should be to strike a balance between these two quality metrics.

5.2 Sample document and result

In this section, the result of entity disambiguation of a sample document is presented and its quality is evaluated based on the aforementioned metrics.

5.2.1 Result of entity recognition and disambiguation

As forecast, a West Indies comeback gave England nightmares at Kensington Oval, but the much-heralded return of Chris Gayle was overshadowed by Marlon Samuels. Samuels is the Yohan Blake to Gayle's Usain Bolt, the silver-medallist support act, but his unbeaten 69 from 46 balls was a reminder ahead of the World T20 that both batsmen possess a considerable threat. Samuels, a vital component of West Indies World T20 victory in Sri Lanka 18 months ago, spoke at the interval of getting fit in the gym, and with lots of running. As ever with Gayle, his expression hidden behind shades, one imagined that his fitness levels had been explored more languidly. It was no surprise that their joint presence persuaded West Indies' captain Darren Sammy to announce that their bid to retain their World T20 crown in Bangladesh started here.

The phrases that were recognized in this document are highlighted using colored boxes. The darkness of the color directly relates to the confidence level expressed by the application for the phrase. A darker color means more confidence in the entity linking, whereas a light color indicates lower confidence.

5.2.1.1 Analysis of the result

The ambiguity tree for this text is shown in figure 4.3. While the tool has made most of the entity linking decisions correctly for the given text, a human reader would have liked to see the phrase *World T20* recognized, as it has some significance attached to it in the context of the article. While, the phrase *getting fit* was linked to the correct entity, namely, *Physical fitness*, there is probably no significance attached to it in the context of this article. Similarly the word *Bangladesh* was incorrectly linked to the entity *Bangladesh National Cricket Team*, while in the context, it was referring to the country. The reason for this behaviour was already explained in the previous chapter. The quality of the result for this piece of text is discussed below.

$$\text{Precision, } \mathfrak{P} = \frac{11}{11+2} = 84.61\% \quad ; \quad \text{Recall, } \mathfrak{R} = \frac{11}{11+1} = 91.6\%$$

The quality metric suggests that the tool has done a fairly reasonable job of identifying and correctly linking the significant phrases in the document (high *precision*), and that it has not left too many significant phrases unrecognized (high *recall*).

Additionally, it is evident from the ambiguity tree shown in figure 4.3, that the phrases had several contending entities and that the tool was able to identify the most appropriate of them.

5.2.2 Results summary

Table 5.1 provides a summary of the results of entity disambiguation, for input documents belonging to different categories mentioned in the first column of the table, for which the contents were randomly chosen from the Internet. The articles that were used are hyperlinked to, which could be followed from the electronic version of this document.

TABLE 5.1: Summary of the results

Category	Words #	$\mathbb{P}_r$	$\mathbb{P}_i$	$\mathbb{P}_u$	$\mathfrak{P}(\%)$	$\mathfrak{R}(\%)$
Nat. Geo.	665	14	0	2	100	87.5
Sports (Cricket)	768	18	2	2	90	90
Environment	795	31	4	2	88.5	93.9
News	249	15	4	2	78.9	88.2
Technology	408	23	6	3	79.3	88.4
Total	2885	101	16	11	**86.3**	**90.1**

For the evaluation, the articles were completely randomly chosen without any affinity towards certain kinds of articles. Additionally, choosing the articles from news channels and from other articles across the web, for the evaluation of this tool, helps us understand and appreciate its applicability in automating entity recognition and disambiguation in this area.

The results show that the tool performs reasonably well (meaning that, it shows good precision and recall) in one of its biggest application areas - automated entity linking for articles on the web.

Chapter 6

Future Work

There are always improvements that could be made to the tool. This chapter aims to address a few of them, and also discusses how.

6.1 Anterior importance of phrases

Even though the application did a reasonable job of identifying keyphrases and linking them to appropriate entities, it was seen that phrases of less importance were sometimes given higher rank than important phrases. This affects the precision, since those insignificant phrases are more likely to be linked to irrelevant entities as the algorithm makes collective entity linking decision, which also affects its overall decision making ability. Therefore, more research could be done on the phrase ranking method, in order to minimize the recognition of insignificant phrases. Ideally, all significant phrases should be ranked above any insignificant phrase.

Further, the line that separates the set of significant phrases from the insignificant ones should be well defined. This would enable the application to pick only the significant phrases for processing and discard all the insignificant ones.

6.2 Using Freebase's knowledge-graph for EL decisions

It was seen in the example with *brands* in section 2.2 that the Freebase knowledge-graph is not mature, in the sense that, it does not capture the relationship between most of the entities that are interrelated to each other in the real world, even though they exist on Freebase independently. It is observed, as a part of this work, that the knowledge graph is mature for entities originating from the *United States* and for other internationally well known entities and

for entities belonging to popular categories such as movies/music, whereas, many entities of the rest of the word do not have a proper knowledge graph.

However, since Freebase is a very fast-growing knowledge base, in the future, when more mature knowledge graph is available for significant amount of entities under different categories, the method proposed by Zheng et al.[5] could be used for more accurate disambiguation. Essentially, the knowledge graph could be used to augment the entity linking decisions from any existing approach, like the one discussed in this work.

Chapter 7

Conclusion

This thesis presented the task of entity disambiguation, wherein all the independent stages, from creating a knowledge-base, to ranking the phrases, to the collective entity linking algorithm that performs disambiguation, were discussed. An application that performs entity recognition and disambiguation was also presented. While we saw that the tool performed very well (with Precision , Recall consistently $> 75\%$) for disambiguating articles across the web, the shortcomings of the tool was also addressed. It was also discussed about the tremendous scope for improvement of this tool, once Freebase as a knowledge-base matures, in the sense that, it starts having more mature knowledge-graph for most of the entities of the world, unlike what it currently is - for only entities related to popular categories, like movies or music, for example.

Appendix A

Computing the n-grams

In was mentioned earlier that n-grams of up to 10 words are computed from the input document, and those that are not present in the keyphrase vocabulary are discarded. An n-gram of words is a contiguous sequence of n words in a given text. For example, for the text *The quick brown fox jumps over the lazy dog*, all its 4-grams are listed below.

The quick brown fox
quick brown fox jumps
brown fox jumps over
fox jumps over the
jumps over the lazy
over the lazy dog

The total number of n-grams in a text with M number of words, is $M - n + 1$. A 1-gram of a given text, is simply the list of all words in the text. For the input document, n-grams for each value of n from 1 to 10, are computed.

Appendix B

$tf \times idf$ score

The **term frequency** $\times$ **inverse document frequency** score is a measure of the importance of a word for a given document in a collection.

Term Frequency. The term frequency of a word w in a document d is simply the number of times w occurrs in d.

Document Frequency. Given a collection C containing a set of documents, the document frequency of a word w in C, written as $\mathrm{DF}(w, C)$, is the total number of documents in C, in which w occurs. Mathematically,

$$\mathrm{DF}(w, C) = |\{d | d \in C \wedge w \in d\}|$$

Inverse Document Frequency. The inverse document frequency of a word w in a collection C, is defined as,

$$\mathrm{IDF}(w, C) = \log_2 \left(\frac{|C|}{\mathrm{DF}(w, C)} \right)$$

The $tf \times idf$ score of a word, is the product of its term frequency and inverse document frequency.

Bibliography

[1] Xianpei Han, Le Sun, and Jun Zhao. Collective entity linking in web text: a graph-based method. pages 765–774, 2011. doi: 10.1145/2009916.2010019. URL http://doi.acm.org/10.1145/2009916.2010019.

[2] Wikipedia. Double redirects — Wikipedia, The Free Encyclopedia. . URL http://en.wikipedia.org/wiki/Wikipedia:Double_redirects.

[3] Wikipedia. Free links — Wikipedia, The Free Encyclopedia. . URL http://en.wikipedia.org/wiki/Wikipedia:Free_links#Free_links.

[4] Wikipedia. Freebase — Wikipedia, The Free Encyclopedia. . URL http://en.wikipedia.org/wiki/Freebase.

[5] Zhicheng Zheng, Xiance Si, Fangtao Li, Edward Y. Chang, and Xiaoyan Zhu. Entity disambiguation with freebase. pages 82–89, 2012. URL http://dl.acm.org/citation.cfm?id=2457524.2457667.

[6] Autocar India. Mahindra xuv500 hits another sales milestone. URL http://www.autocarindia.com/auto-news/mahindra-xuv500-hits-another-sales-milestone-369053.aspx.

[7] Rada Mihalcea and Andras Csomai. Wikify!: linking documents to encyclopedic knowledge. pages 233–242, 2007. doi: 10.1145/1321440.1321475. URL http://doi.acm.org/10.1145/1321440.1321475.

[8] Ted Pedersen, Amruta Purandare, and Anagha Kulkarni. Name discrimination by clustering similar contexts. pages 226–237, 2005. doi: 10.1007/978-3-540-30586-6_24. URL http://dx.doi.org/10.1007/978-3-540-30586-6_24.

[9] David Milne and Ian H. Witten. Learning to link with wikipedia. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management*, CIKM '08, pages 509–518, New York, NY, USA, 2008. ACM. ISBN 978-1-59593-991-3. doi: 10.1145/1458082.1458150. URL http://doi.acm.org/10.1145/1458082.1458150.

[10] Hanghang Tong, Christos Faloutsos, and Jia-Yu Pan. Fast random walk with restart and its applications. In *Proceedings of the Sixth International Conference on Data Mining*, ICDM '06, pages 613–622, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 0-7695-2701-9. doi: 10.1109/ICDM.2006.70. URL http://dx.doi.org/10.1109/ICDM.2006.70.