Location Tagging in Text

by

Shawn Brunsting

A research paper
presented to the University of Waterloo
in partial fulfilment of the
requirement for the degree of
Master of Mathematics
in
Computational Mathematics

Supervisor: Prof. Hans De Sterck

Waterloo, Ontario, Canada, 2015

I hereby declare that I am the sole author of this report. This is a true copy of the report, including any required final revisions, as accepted by my examiners.

I understand that my report may be made electronically available to the public.

Abstract

Location tagging, also known as geotagging, is the process of assigning geographical coordinates to input data. In this project we present an algorithm for location tagging text.

Our algorithm makes use of previous work in natural language processing by using a state-of-the-art part-of-speech tagger and named entity recognizer to find blocks of text which may refer to locations. A knowledge base is then used to find a list of possible locations for each of these blocks of text. Finally, one location is chosen for each block of text by assigning distance-based scores to each location and repeatedly selecting the location and block of text with the best score.

We tested our location tagging algorithm with geotagged Wikipedia articles, where classification approaches have achieved median errors as low as 11 km [5]. However, the maximum accuracy of these approaches is limited by the class size, so future work may not yield significant improvement.

Our algorithm tags a location to each block of text that was identified as a possible location reference, meaning one text can have multiple tags. When we considered only the tag with the highest distance-based score, we achieved a 10th percentile error of 490 metres and median error of 54 kilometres. When we considered the five location tags with the greatest scores, we found that 50% of articles were assigned at least one tag within 8.5 kilometres of the article's true location. Other classification-based approaches may be reaching their upper limit for accuracy, but our precision-focused approach has high accuracy for some articles and shows significant potential for improvement overall.

Acknowledgements

Dedication

This paper is dedicated to my wife Alanna, whom I had the honour of marrying while completing this project. She generously prepared many delicious snacks, meals, and cups of tea to keep me going throughout the programming and writing for this project.

Table of Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

This paper explores the problem of extracting location information from text. The goal is to assign high precision geographical coordinates to places that are mentioned in texts from various sources.

This chapter discusses some background information and previous work in the area of location tagging. In the literature this is also referred to as geolocation. Chapter 2 describes in detail the algorithm that we developed to solve this problem. We test this algorithm using geotagged Wikipedia articles in Chapter 3. Finally, we summarize the results and discuss some future work in Chapter 4.

Appendix A lists the external software that was used in the implementation of this project, and Appendix B describes how the test data for Chapter 3 was obtained.

1.1   Previous Work in Classification Approaches

Most studies in location tagging formulate it as a classification problem and use various machine learning approaches to solve it.

Classification problems begin with a defined set of classes. In location tagging, these classes can take many forms including cities, countries, or areas within a range of latitude and longitude coordinates. The goal of the classification problem is to assign the correct class to each text.

Wing and Baldridge created classes using simple geodesic grids of varying sizes [9]. For example, one of their grids divided the surface of the Earth into 1° by 1° regions.

1

Each of these regions was a class for their classification problem. They compared different models on various grid sizes, and tested these models using geotagged Wikipedia articles. They measured the distance between the article's true location and the location that was predicted by the model, and their best model achieved a median distance of 11.8 km.

Roller et al. realized that defining classes using a uniform latitude and longitude grid can be sensitive to the distribution of documents across the Earth [5]. They looked at building an adaptive grid, which attempts to define classes such that each class has a similar number of training documents. They also tested their models using geotagged Wikipedia articles, and found a median error of 11.0 km. This is an improvement over the previous work [9].

Han et al. focused on location tagging of Twitter messages [4]. They attempted to find the primary home location of a user by assembling all their tweets into a single document, which was then used as input to their models. Their classes were major cities along with their surrounding area. Their best model that only used the text of the tweet obtained a median error of 170 km. They obtained much greater accuracy when they incorporated additional information such as the home location on the user's profile, but this type of data is not available for general text geolocation.

One of the major challenges with Twitter messages is their unstructured nature. Tweets often contain spelling errors, abbreviations, and grammar mistakes which can make them difficult to interpret. Furthermore, some early work on this project discovered that most geotagged tweets contain little geographic information in their text. This means that there is a very low threshold for the maximum accuracy we can expect to achieve when attempting to apply any location tagging algorithm to tweets. For this reason accuracy tests with tweets are left for future work.

1.2 Drawbacks of Classification Approaches

It was decided that formulating our problem as a classification problem would not be feasible for this project, as our goal was to obtain high precision. For example, if a text mentions the CN Tower then we want to return the coordinates of that building, rather than the coordinates for the city of Toronto where it is located. Formulating this as a classification problem with this level of precision would require defining a class for every named location in the world. Furthermore, to apply these machine learning approaches we would need to have a large training set, ideally with multiple sample texts for each class. Obtaining this training data would be difficult, and even if the data was available it

would likely be computationally infeasible to train such a model without significant time and computing resources.

In general, we did not want to use any approach that relies heavily on training data, as we want our location tagger to be as general as possible. A tagger trained with Wikipedia articles might show worse performance when given other types of text, such as news articles or tweets. Acquiring good training data that is representative of all the types of texts we want to geolocate would be very difficult.

Instead we abandoned the classification approaches and turned towards natural language processing.

1.3   Natural Language Processing

Natural language processing (NLP) encompasses many tasks that are related to understanding human language. In this project we wish to understand location references in text, so it is natural to apply NLP techniques to this problem.

1.3.1   Part of Speech Tagging

Part-of-speech (POS) tagging is the process of assigning part-of-speech tags to words in a text. The tagset may vary by language and the type of text, but typically includes tags such as nouns, verbs, and adjectives. Assigning these tags is an important step towards understanding text in many NLP tasks.

The Stanford Natural Language Processing Group provides a state-of-the-art POS tagger based on the research of Toutanova et al. [8, 7]. They use various models that observe features of words in the text (such as capitalization, for example) and predict the most likely part-of-speech tag for each word. They train these models using a large set of annotated articles from the Wall Street Journal.

Their software has two main types of models. One of these types is called the left-3-words model. The tag that is applied to each word in this model is influenced by the tags assigned to some of the previous words in the sentence.

The other type of model is known as the bidirectional model [7]. Tags in these models are influenced by tags on both sides of the word, not just the previous words. This makes these models more complex. They have a slight accuracy improvement over the left-3-words models, but at the cost of an order of magnitude longer running time.

In this project we use Stanford's POS tagger with one of their pre-trained left-3-words models. How our algorithm uses this software will be discussed in Section 2.3.

1.3.2   Named Entity Recognition

A Named Entity Recognizer (NER) finds proper nouns in text and determines to what type of entity the noun refers. This will be a valuable tool in our approach to location tagging, as we are primarily looking for locations that are named entities in the text.

The Stanford Natural Language Processing Group also provides an NER based on the work of Finkel et al. [2]. Their software uses a conditional random field model to identify different classes of named entities in text. Their pre-trained model that was used in this project attempts to identify each named entity as a location, person, or organization. In Section 2.3 our algorithm will use entities that are identified as locations by this software.

1.3.3   Named Entity Disambiguation

After completing an NER task there is often some ambiguity. For example, if the NER software determines that some text mentions a location called *London*, does it refer to a city in the UK or a city in Ontario? Choosing the correct interpretation is called disambiguation.

Habib explored both the extraction and disambiguation steps for named entity recognition, along with the link between these steps [3]. He discovered that there is a feedback loop, where results from the disambiguation step can be used to improve the extraction step. This occurs because the disambiguation step can help to identify false positives, that is, words that were identified as a named entity by the extraction step but are not true named entities.

Named entities which refer to locations are called *toponyms*, and a major portion of Habib's [3] work discusses the extraction and disambiguation of toponyms in semi-formal text. He disambiguated toponymns in vacation property descriptions to determine in which country the property was located. While country-level locations do not provide the level of precision we desire for this project, his work served as inspiration for many steps of the algorithm we present in Chapter 2.

Chapter 2

Algorithm

Given a segment of text, we want to find locations that are mentioned in the text. For example, in a text that states "Bob drove from Waterloo to Toronto" we want to find the names *Waterloo* and *Toronto* and determine which locations are meant by those words. Does *Waterloo* mean a city in Ontario, Iowa, Belgium, or somewhere else? The mention of *Toronto*, which can refer to another city in Ontario, suggests that the correct answer is Waterloo, Ontario. This type of reasoning was developed into an algorithm which is formally described in this chapter.

Section 2.1 gives an overview of the algorithm. Section 2.2 formally defines the terminology we will use in the rest of the paper. Section 2.3 describes how names like *Waterloo* and *Toronto* are extracted from the text using part-of-speech tagging and named entity recognition. Section 2.4 describes how we discover that geographic names like *Waterloo* can refer to multiple locations (e.g., Ontario, Iowa, or Belgium) by searching a knowledge base. Section 2.5 describes how we disambiguate between the possible locations for each name (e.g., determine that *Waterloo* and *Toronto* refer to cities in Ontario). Finally, the algorithm is summarized in Section 2.6.

Much of this algorithm was developed with particular examples in mind. Looking at these examples gave insights into the information that is available for our algorithm to use. The above example with *Waterloo* and *Toronto* will be revisited and explained in more detail in Section 2.2.1. Another development example will be used in Sections 2.3.5, 2.4.1, and 2.5.5 to summarize the details for each step.

2.1 Overview

The geotagging algorithm that we developed follows a number of steps. Each step contains many subtleties, so for now we start with a general overview which we call Algorithm 1.

Algorithm 1 Simplified Overview of Tagging Algorithm
___
1: Extract potential location references from the text. This is described in detail in Section 2.3.
2: Search for each potential location reference in a knowledge base. This will give a list of locations that are possible matches for the reference. This is described in Section 2.4.
3: For each potential location reference, determine to which of the knowledge base matches it most likely refers. This is called disambiguation, and is described in detail in Section 2.5.

Section 2.2 will define some terminology which will allow us to write Algorithm 1 more precisely. Section 2.6 will summarize this chapter to give a more detailed version of Algorithm 1.

2.2 Terminology

Before we continue with our description of the algorithm, we need to precisely define our terminology:

- A **phrase** is a word or sequence of adjacent words in the text. In our implementation a phrase is stored as a single string. We let P be the set of unique phrases that we consider to be possible location references in a text.

- A **term** is an object associated with a phrase. This object includes some metadata, such as the position of the phrase in the text. Let T be the set of terms that we consider to be possible location references in the text. Note that if a phrase occurs multiple times in the text, there will be one term for each occurrence. Therefore $|P| \leq |T|$. For some $t \in T$, let t_{phr} be the phrase associated with the term.

- A **result** is a single location that is listed in the knowledge base. For each phrase $p \in P$ we have a set of results R^p. For each result $r \in R^p$, the name of result r is similar to the phrase p ("similarity" is defined by the knowledge base software we use in Section 2.4). As a shorthand, let $R^t \equiv R^{t_{phr}} \ \forall t \in T$.

- For a term $t \in T$ and a result $r \in R^t$, we define the **score** of the result to be S_r^t. This score will be used in the disambiguation step.

- For two terms $t_1, t_2 \in T$ we define $W_{t_2}^{t_1}$ to be the **weight** of term t_2 when we assume that term t_1 is a true reference in the text. Weights are used to reduce bias in some score calculations when we find terms in T that conflict with each other. Section 2.5.2 will precisely define conflicts and weights.

We can now give an overview of the algorithm using this terminology. Algorithm 2 is a re-writing of Algorithm 1 using the terms defined above.

Algorithm 2 Technical Overview of Tagging Algorithm

1: Find all terms in the text that are potential location references. This is done using part-of-speech tagging and named entity recognition, and gives us the sets T and P.
2: For each phrase $p \in P$, use p as a search query in the knowledge base. The results of the query are the set R^p.
3: Reduce the set T to remove terms which conflict with each other. Update the set P accordingly to reflect changes in T. For each phrase $p \in P$, match p to a single result $r \in R^p$. This is done by assigning distance-based scores to each result, and selecting results with the greatest scores.

In step 3 of Algorithm 2 we can see that each phrase, rather than each term, is matched to a single result. This means all terms with the same phrase are assumed to refer to the same location. This assumption lowers the computational complexity of the algorithm and simplifies its implementation.

For example, a text with two occurrences of *Waterloo* will assume both terms refer to the same location. However, terms with phrases *Waterloo Ontario* and *Waterloo Belgium* will not be assumed to refer to the same location as they have different phrases.

2.2.1   Simple Example

In this section we revisit the "Bob drove from Waterloo to Toronto" example that was presented at the beginning of this chapter. We can now describe how our algorithm finds location tags for this example using our terminology and Algorithm 2.

In step 1 of Algorithm 2 we use NLP tools (in particular, part-of-speech tagging and named entity recognition) to identify the words *Waterloo* and *Toronto* as possible locations. *Waterloo* and *Toronto* are phrases, and we will have one term in T for each of these phrases.

In step 2 of this algorithm we search a knowledge base for both *Waterloo* and *Toronto*. We discover cities in Ontario, Iowa, and Belgium with the name *Waterloo*. Each of these cities is a result. Similarly, for *Toronto* we discover a city in Ontario and a city in Ohio.

In this example there are no conflicts between terms. So in step 3 we simply need to choose which of the results to use for each phrase. Weights have no effect on the score calculations in this example.

We calculate scores for each result based on the distance to other results. Waterloo, Belgium has the worst score because it is very far away from all results for *Toronto*. Waterloo, Ontario and Toronto, Ontario have the best scores because each one is close to a result for the other term. Our algorithm picks one of these as a true location in the text. Let's say Toronto, Ontario is chosen. Then the result in Ohio is discarded, and scores are recalculated. Now Waterloo, Ontario has the best score out of all results for terms that still have multiple results, because it is closest to the *Toronto* that we chose in Ontario. So we choose Waterloo, Ontario and the other two results for *Waterloo* are discarded.

Our algorithm can now terminate, as it has chosen one location for *Waterloo* and one location for *Toronto*. It returns two cities: Waterloo, Ontario and Toronto, Ontario.

2.3   Location Extraction

The extraction phase of the algorithm uses the part-of-speech (POS) tagger and named entity recognizer (NER) from Stanford that were described in Section 1.3. The entire text is tagged by both of these models. Based on the output of these taggers, the algorithm creates the set of terms T which represents all potential location references in the text.

Neither the POS tagger nor the NER are perfect. If they were, then we could simply use the NER to find all locations in the text and this step of the algorithm would be complete. Instead, this algorithm supplements the NER tags with the POS tagger.

2.3.1   Tagging the Text

The POS tagger from Stanford assigns tags to each word in the text. The full list of possible tags is given in [6], but our algorithm only looks for a subset of tags which are relevant to our approach. Some of these tags are grouped together and considered to be equivalent by our algorithm. The tags used and how they are grouped are described in Table 2.1.

POS Tag	POS Tag Description	Our Grouping
CC	Coordinating conjunction	conjunction
CD	Cardinal number	adjective
IN	Preposition or subordinating conjunction	preposition
JJ	Adjective	adjective
NN	Noun, singular or mass	noun
NNS	Noun, plural	noun
NNP	Proper noun, singular	noun
NNPS	Proper noun, plural	noun
TO	to	preposition

Table 2.1: Part of Speech Tags Used in the Tagging Algorithm

The NER tags each word in the text with one of four possibilities: LOCATION, PERSON, ORGANIZATION, or O (meaning "other"). In the discussion in Section 2.3.2 any word with a LOCATION tag from the NER is considered equivalent to a word with a noun tag from the POS tagger. The distinction between LOCATIONS and nouns becomes important in Section 2.3.3.

2.3.2 Extracting Terms

After each word in the text has been tagged by both the POS tagger and the NER, we build our set of terms T which holds all potential location references. Any phrase in the text is considered a potential location reference if it satisfies two properties:

1. The phrase contains at least one word with a noun tag from the POS tagger or a LOCATION tag from the NER.

2. All words in the phrase are tagged with a noun or adjective tag from Table 2.1, or a LOCATION tag from the NER.

The details about why these properties were chosen is described in the rest of this section. Note that for the remainder of this section, a *noun* refers to any word that is tagged with LOCATION by the NER, or with one of the noun tags in Table 2.1 by the POS tagger.

Some locations, such as *New York*, have multiple words in their name. Each word is tagged individually by the POS tagger and NER, so we need to consider these multi-word possibilities when building T.

If multiple nouns occur adjacent to each other in the text, then we do not know if these nouns refer to multiple locations or to one location. So we add all possibilities to the set T, and we will resolve conflicts in Section 2.5.

For example, if the algorithm discovers the phrase "New York City" in the text, with each of the three words tagged as a noun, then the algorithm would add six terms to the set T: *New*, *York*, *City*, *New York*, *York City*, and *New York City*.

During the development of our algorithm it was discovered that some location references contain words tagged as adjectives. For example, given the text "georgian college" the POS tagger decides that *georgian* is an adjective and that *college* is a noun. *georgian* is part of the name, even though it was not tagged as a noun. So our algorithm was modified to consider adjectives when they are part of a multi-word phrase with other nouns. However, it does not consider phrases that only contain adjectives (otherwise we could simply add adjectives to the noun group in Table 2.1). In the "georgian college" example, both *college* and *georgian college* are added to T.

If a text contains a street address, we want the street number to be part of the term. This will allow for greater precision in finding the proper location. Numbers are treated the same way as adjectives for this reason. So the example text "200 University Avenue" would generate five terms: *University*, *Avenue*, *University Avenue*, *200 University*, and *200 University Avenue*.

Note that our set T is not finalized yet. Some of the terms we added in this section may be removed, as described next in Section 2.3.3.

2.3.3   Removing Terms

In Section 2.3.2 we built a large set T of terms which could be location references in the text. In this section we describe how T is filtered.

The amount of time required for the knowledge base searches in Section 2.4 grows linearly with $|P|$, and the amount of time required for the disambiguation in Section 2.5 is cubic in $|T|$. Furthermore, including too many terms in T which are not true location references may make it difficult to disambiguate properly. Therefore it is advantageous to keep the size of T small.

The first step in reducing T is to check whether any of the terms contain words that were tagged as locations by the NER model. If there are LOCATION tags for any word in any term in T, then we keep only terms which contain at least one word tagged with LOCATION. All others are removed from the set T.

If no words were tagged with LOCATION, then the algorithm must rely solely on the results of the POS tagger. However, the set of nouns in a text can be quite large, so we still wish to filter T.

In the case where no LOCATION tags were found, the next step is to look for terms that occurred after prepositions. Many prepositions describe spatial relationships, so they can be strong indicators that a term does refer to a location. For example, the text "Bob travelled from Waterloo" contains the preposition *from*. In this case, the preposition tells us that *Waterloo* is a location, and that *Bob* departed there.

However, prepositions can describe not only spatial relationships, but also temporal ones. For example, "Bob lived there for five years" uses *for* as a temporal preposition. This type of preposition should not be included in our algorithm. The implementation that was used in this project explicitly ignored the word *for* as a preposition, as this was observed to increase accuracy for some development examples. Future work could refine this list to exclude more non-spatial prepositions.

If the text contains terms that occurred after prepositions but no words tagged with LOCATION, then these terms are retained while all others are discarded from T. A term is considered to be after a preposition if all words between the preposition and the term are tagged with any of the tags in Table 2.1. It is for this reason that conjunctions are included in that table. Including conjunctions ensures that a text such as "Guests travelled from Waterloo and Toronto" will consider both Waterloo and Toronto.

If the text contains no LOCATION tags and there are no terms that follow prepositions, then no terms are removed from T.

2.3.4 Postal Codes

Some of the development example texts contained postal codes. Postal codes can give very precise location information, so regular expressions were used to find postal codes that match the formats used by Canada, the United States, and the Netherlands. (The Netherlands were included because Spotzi, a partner company for this project, is based there.) All occurrences of postal codes are added to the set T if they are not already in the set due to previous steps. This occurs after the filtering in Section 2.3.3.

2.3.5 Example

To demonstrate how the extraction step works, we will walk through an example in this section. This short example was used in the development of this algorithm, and shows

11

Word	NER Tag	POS Tag	POS Tag Description
A	O	DT	Determiner
beautifull	O	NN	Noun, singular or mass
clean	O	JJ	Adjective
house	O	NN	Noun, singular or mass
for	O	IN	Preposition or subordinating conjunction
rent	O	NN	Noun, singular or mass
,	O	,	
Walking	O	VBG	Verb, gerund or present participle
distance	O	NN	Noun, singular or mass
to	O	TO	to
RVH	O	NN	Noun, singular or mass
and	O	CC	Coordinating conjunction
Georgian	O	JJ	Adjective
college	O	NN	Noun, singular or mass

Table 2.2: Part of Speech Tags for Example Text

many of the cases that were described in the previous sections. We will continue using this example in Sections 2.4.1 and 2.5.5.

The text is from a Kijiji listing, and consists of a single sentence: "A beautifull clean house for rent, Walking distance to RVH and Georgian college." This text is not properly structured. In particular, it has a spelling error and some improper capitalization. This makes the text challenging.

First we use the NER and POS tagger to tag all words in the text. The full list of tags for this example is given in Table 2.2. We can see that the NER assigns no LOCATION tags in this text, so we must rely solely on the POS tags.

Next, we start building our set T. We take all nouns, along with adjectives that are adjacent to them. So our set P that corresponds to T is:

$$P = \{beautifull, beautifull\ clean, beautifull\ clean\ house, clean\ house, house,$$
$$rent, distance, RVH, Georgian\ college, college\}$$

Now we start to reduce this set. We have no terms that contain words tagged with LOCATION, but we do have terms that occur after prepositions.

The terms with phrases *beautifull*, *beautifull clean*, *beautifull clean house*, *clean house*, *house*, and *distance* are all discarded because they do not occur after a preposition. The

12

term with *rent* occurs after a preposition, but as described in Section 2.3.3 this preposition is the word *for* and is assumed to not be a spatial preposition. So *rent* is also discarded.

The preposition *to* occurs before *RVH*, so the term for *RVH* remains in T. Similarly, *Georgian college* and *college* are considered to occur after this preposition, because they are separated from this preposition only by tags in Table 2.1.

So our set P that corresponds to our final set T is:

$$P = \{RVH, Georgian\ college, college\}$$

This set is used in the next step of the algorithm.

2.4   Searching a Knowledge Base

After we generate our set T (and the set P that corresponds to it), the next step is to search for each $p \in P$ in a knowledge base. OpenStreetMap is used as the knowledge base for this project. The OpenStreetMap data is queried using a tool called Nominatim (https://nominatim.openstreetmap.org/).

OpenStreetMap is a database that contains mapping data for the entire globe, including roadways, cities, and points of interest. This data is created and corrected by a large community of contributors, and is freely available.

Nominatim is an open-source tool that allows users to search the OpenStreetMap data. Using clever indexing and a large amount of system resources, Nominatim allows queries of the vast OpenStreetMap data to be completed in seconds. [1]

Nominatim allows a number of parameters to be specified with the query. For example, searches can be limited to a particular region or country. One can also specify the maximum number of search results that should be returned for each query. In this paper the maximum number was set to 10.

---

[1] There are donated servers running Nominatim which are free for light use, but they have a usage policy to prevent users from overloading the server. This usage policy gives a limit of one query per second. Our algorithm requires one query for each $p \in P$, so anyone who wants to use this algorithm extensively should set up their own Nominatim server. The tests in Chapter 3 did use the freely available servers with a delay written into the code to ensure that at least one second passed between each query sent to Nominatim. This meant the tagging algorithm completed much slower than it would otherwise, but it saved time on the engineering effort required to set up a Nominatim server.

For each phrase $p \in P$ we query Nominatim and obtain a set of results R^p, where $0 \leq |R^p| \leq 10$. If $|R^p| = 0$ then we have no results for p, and we discard all terms $t \in T$ that satisfy $t_{phr} = p$.

If $|R^p| \geq 1$ then we use these results in the disambiguation step (step 3 of Algorithm 2), which selects for each phrase p the correct result in R^p. Each result $r \in R^p$ contains latitude and longitude coordinates which are used to calculate distances between results as described in Section 2.5.1.

Another field that is included with each Nominatim search result is called "importance". This field is not well-documented, but after reading the source code it appears that a number of different factors are used to calculate this importance. These factors include:

- The type of location (building, city, country, etc.)

- The PageRank of the Wikipedia article about the location (if applicable)

- The string similarity between the query string and the name of the location

The importance field is used as a tie-breaker in some steps of the disambiguation algorithm in Section 2.5. It is also used by Nominatim to sort results. If Nominatim finds more than 10 results for a query, it will return only the 10 most important results.

2.4.1 Example

Here we continue the example from Section 2.3.5. When we last saw this example we had three terms, and their corresponding phrases were

$$P = \{RVH, Georgian\ college, college\}$$

We use each phrase as a search query with Nominatim. However, for this example we will limit Nominatim to return a maximum of three results for each phrase (instead of 10, which was used for the rest of this project).

The results for each query are given in Table 2.3. Nominatim found only one result for *RVH*, two for *Georgian college*, and three for *college*. The phrase *college* would have more results if we increased the result limit.

Query	Results				
	#	Name	Address	Latitude	Longitude
RVH	1	Royal Victoria Regional Health Centre	201, Georgian Drive, Barrie, Ontario, Canada	44.41476385	-79.663043119369
Georgian college	1	Georgian College	Georgian Drive, Barrie, Ontario, Canada	44.4127259	-79.6710631228231
	2	Georgian College	Raglan Street, Collingwood, Ontario, L9Y3J4, Canada	44.48446015	-80.1917274556815
college	1	College	Yonge Street, Church-Wellesley Village, Toronto, Ontario, M5B 2H4, Canada	43.6613331	-79.3830661
	2	College	Fairbanks North Star, Alaska, United States of America	64.8569444	-147.8027778
	3	College	Los Baños, Laguna, Calabarzon, Philippines	14.1624278	121.2409175

Table 2.3: Example Nominatim Results

2.5 Disambiguation

The disambiguation step of the algorithm chooses the best result for each phrase in the text. It does this by assigning scores to each result. Multiple scoring functions were tested, and each one is described in Section 2.5.3. Most of these scoring functions use weights that are assigned to each term, and these weights are described separately in Section 2.5.2. Section 2.5.4 describes two algorithms that use the scores to perform the disambiguation.

2.5.1 Distance

The key to the disambiguation step in our algorithm is the physical distance between results. For example, suppose a text mentions *London*. Does this mean London, UK or London, Ontario? Clues can often be found in the rest of the text. If the text also mentions *Toronto, Ontario*, for example, then this is a good indication that London was meant to refer to the city that is also in Ontario.

To calculate the distance between two results, we use the following formula. Let a and b be two search results from Nominatim. Let a_x and b_x be their longitude coordinates, and let a_y and b_y be their latitude coordinates, respectively. Nominatim gives the coordinates in degrees, but for this discussion we will assume they have been converted to radians. The distance between a and b is then calculated with Equation 2.1.

$$d(a, b) = 6371 \arccos\left(\sin(a_y)\sin(b_y) + \cos(a_x)\cos(b_x)\cos(a_x - b_x)\right) \qquad (2.1)$$

Equation 2.1 is the great circle distance on a sphere, where 6371 is the mean radius of the Earth in kilometres. The Earth is not a perfect sphere, but this gives a good approximation for the distance between the two results.

2.5.2 Weights

Weights are used when there are terms in the set T which conflict with each other. Terms conflict when they overlap. This is best explained with the New York City example that was presented in Section 2.3.2.

Six terms were added to the set T when the extraction step found the nouns *New*, *York*, and *City* adjacent to each other in the text: *New*, *York*, *City*, *New York*, *York City*, and *New York City*. The terms *York* and *New York* conflict with each other because the text

Term	Conflicting Terms
New	*New York, New York City*
York	*New York, York City, New York City*
City	*York City, New York City*
New York	*New, York, York City, New York City*
York City	*York, City, New York, New York City*
New York City	*New, York, City, New York, York City*

Table 2.4: Conflicting Terms in "New York City"

could not have meant to refer to a location called *York* and a different location called *New York*. This is because the terms *York* and *New York* overlap. The full list of conflicting terms for this example is given in Table 2.4.

The purpose of the weights we will define in this section is to properly account for conflicting terms in the disambiguation step. The scoring functions we will define in Section 2.5.3 are based on the total distance to other terms in the text. For this New York City example, the text might only refer to one location in that segment of text. However, we have six terms in T to represent that piece of text. Assigning weights to these six terms will allow us to reduce bias in the scoring functions until we can determine which interpretation of that segment is correct.

We can think of the weight $W_{t_2}^{t_1}$ as the probability that term t_2 is a true location reference in the text given that t_1 is a true location reference. The weights we calculate are heuristically defined and do not correspond to true probabilities, but they obey many of the same properties as probabilities.

Weights are defined such that $0 \leq W_{t_2}^{t_1} \leq 1 \ \forall t_1, t_2 \in T$. If no terms in T conflict with each other then all weights are equal to 1. The remainder of this section describes how weights are calculated when some terms do conflict.

We begin our mathematical definition of weight with the conflicting case:

$$W_{t_2}^{t_1} = 0 \quad \forall t_1, t_2 \in T | t_1 \text{ and } t_2 \text{ conflict} \tag{2.2}$$

Equation 2.2 comes from our previous definition of weights: $W_{t_2}^{t_1}$ is the weight of term t_2 when we assume that t_1 is in the text. Thinking of the weight in terms of probabilities, if t_1 is truly referenced in the text, then any terms that conflict with t_1 cannot be truly referenced in the text. So when t_1 and t_2 conflict, the weight $W_{t_2}^{t_1}$ is zero.

Note that a term does not conflict with itself. Thinking of the weights as the probabilities we described earlier, we can define:

$$W_t^t = 1 \quad \forall t \in T \tag{2.3}$$

Before we define the weight in the other cases, we need another definition. Conflicting terms result in *groups*. Removing a term in a group will affect the weights of all other terms in the group.

We define $G(t)$ to be the group of term t. $G(t)$ is the smallest set of terms that satisfies the following properties:

$$t \in G(t) \quad \forall t \in T$$
$$G(t_1) = G(t_2) \quad \forall t_1, t_2 \in T | t_1, t_2 \text{ conflict}$$

This definition implies that

$$G(t_1) = \{t_1\} \quad \forall t_1 \in T | t_1 \text{ does not conflict with any } t_2 \in T$$

Using this definition, we can conclude that all terms in Table 2.4 are in the same group.

Groups have a number of different interpretations depending on the amount of overlap. An interpretation is a subset of a group which contains terms that do not conflict. An interpretation does not have to cover all words in the segment of text, but it must contain enough terms such that no non-conflicting terms can be added to the group.[2] Table 2.5 lists all four interpretations for the New York City example.

Next we will define the weight $W_{t_2}^{t_1}$ for the case where $t_1 \notin G(t_2)$. It is given by:

$$W_{t_2}^{t_1} = \frac{1}{\# \text{ of interp. of } G(t_2)} \sum_{\substack{\text{interp. of } G(t_2) \\ \text{that contain } t_2}} \frac{1}{\# \text{ of terms in this interp.}} \tag{2.4}$$

The weights given to each term in each interpretation are included in the last column of Table 2.5.

Table 2.6 continues the calculations from Table 2.5 to give the weights $W_{t_2}^{t_1}$ for the case where $t_1 \notin G(t_2)$ and t_2 is a term in the New York City group. For example, this table tells us that when $t_1 \notin G(New)$ then $W_{New}^{t_1} = \frac{5}{24}$.

---

[2]In our New York City example all our interpretations cover all words in that segment of text. However, if the term *New* was missing from T because it was tagged as an adjective, then {*York*, *City*} and {*New York*, *City*} would both be valid interpretations. However, {*York*} would not be valid, since the term for *City* can still be added.

Interpretation Number	Terms In Interpretation	Number of Terms	Weight Given To Each Term
1	*New, York, City*	3	$\frac{1}{4} * \frac{1}{3} = \frac{1}{12}$
2	*New York, City*	2	$\frac{1}{4} * \frac{1}{2} = \frac{1}{8}$
3	*New, York City*	2	$\frac{1}{4} * \frac{1}{2} = \frac{1}{8}$
4	*New York City*	1	$\frac{1}{4} * \frac{1}{1} = \frac{1}{4}$

Table 2.5: Example Interpretations for "New York City"

Term t_2	Weight from Interp. 1	Weight from Interp. 2	Weight from Interp. 3	Weight from Interp. 4	Weight
New	$\frac{1}{12}$	0	$\frac{1}{8}$	0	$\frac{5}{24}$
York	$\frac{1}{12}$	0	0	0	$\frac{1}{12}$
City	$\frac{1}{12}$	$\frac{1}{8}$	0	0	$\frac{5}{24}$
New York	0	$\frac{1}{8}$	0	0	$\frac{1}{8}$
York City	0	0	$\frac{1}{8}$	0	$\frac{1}{8}$
New York City	0	0	0	$\frac{1}{4}$	$\frac{1}{4}$
Interp. Total	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	

Table 2.6: Weights $W_{t_2}^{t_1}$ for Terms in "New York City" where $t_1 \notin G(t_2)$

We can see from Table 2.6 that each interpretation has the same total weight ($\frac{1}{4}$ in this case), and that all weights in a group total to 1. This is true for every group in any set T in our approach. Weights were defined such that these properties were satisfied, but this was a heuristic choice. Future work may discover that a different definition of weights gives better performance.

Finally, we must define $W_{t_2}^{t_1}$ for the case where $t_1 \in G(t_2)$ and t_1 and t_2 do not conflict. As discussed earlier, the weight $W_{t_2}^{t_1}$ is the weight of t_2 when we assume that t_1 is in the text. Therefore, all we need to do to calculate the weight in this case is temporarily remove terms from T that conflict with t_1. This will change the groups $G(t_1)$ and $G(t_2)$ such that t_1 is no longer in $G(t_2)$ (because t_1 no longer conflicts with any terms, $G(t_1) = \{t_1\}$). Our calculation of the weights then proceeds as before with Equation 2.4.

For example, suppose we want to calculate $W_{t_2}^{New}$ for $t_2 \in G(New)$. $W_{New\ York}^{New} = W_{New\ York\ City}^{New} = 0$ because those terms conflict. We remove those terms, so New becomes the only member of its own group. So $W_{New}^{New} = 1$. Finally, the remaining terms have only two interpretations: $\{York, City\}$ and $\{York\ City\}$. When we calculate those weights we find that $W_{York}^{New} = W_{City}^{New} = \frac{1}{4}$ and $W_{York\ City}^{New} = \frac{1}{2}$.

2.5.3 Scoring Functions

There are a few more definitions we need in order to describe the scoring functions that are used in the disambiguation step:

- For two search results r_1 and r_2, let $d(r_1, r_2)$ be the **distance** between them in kilometres. This is calculated using Equation 2.1.

- As a shorthand, for a search result r and a term t we define $c(r, t) = \min_{r' \in R^t} d(r, r')$. Then $c(r, t)$ is the **shortest distance** between result r and any result for t.

The definitions above and in Section 2.2 are now combined in different ways to provide eight unique scoring functions, which are described in detail in the rest of this section. All scoring functions are defined such that greater scores are considered "better." Scoring functions 2.5 and 2.6 are "better" when the scores are closer to zero, because this means there is less total distance to the other terms. Without the minus signs they would always be non-negative, so by adding the minus signs we can still say that the greatest scores are best for all scoring functions.

The first scoring function we consider is called **Total Distance** and is given by Equation 2.5. For a term $t^1 \in T$ and a result $r_1 \in R^{t^1}$, this function simply adds up the minimum

distance between r_1 and the closest result for each other term in T. Note that for any term t such that $t_{phr} = t^1_{phr}$ (including the original term $t = t^1$) we have $c(r_1, t) = 0$. Therefore we do not need to explicitly ensure that $t^2 \neq t^1$ in the summation in Equation 2.5. However, we do explicitly ensure that $W^{t^1}_{t^2} \neq 0$ so we do not consider conflicting terms.

$$S^{t^1}_{r_1} = - \sum_{\substack{t^2 \in T \\ W^{t^1}_{t^2} \neq 0}} c(r_1, t^2) \tag{2.5}$$

The Total Distance score is always negative, because the distances $c(r_1, t^2)$ are always positive then multiplied by -1. The result with the best score would be the one whose score is closest to zero.

The second scoring function we consider is called **Weighted Distance**, given by Equation 2.6. This is equivalent to Equation 2.5 except for the multiplication by the weight term. The reasoning behind these weight terms and how they are calculated is described in Section 2.5.2.

$$S^{t^1}_{r_1} = - \sum_{t^2 \in T} W^{t^1}_{t^2} c(r_1, t^2) \tag{2.6}$$

Scoring functions 2.5 and 2.6 may be sensitive to terms that are extremely far away from the others. This can occur when the location extraction step (step 1 of Algorithm 2) extracts a phrase that does not refer to a location, but does have results in the knowledge base. The location extraction step is not perfect, so this is a common occurrence. The remaining scoring functions attempt to deal with this.

The next two scoring functions we consider are called **Inverse** (Equation 2.7) and **Weighted Inverse** (Equation 2.8). They are similar to Equations 2.5 and 2.6 respectively, except that we use the reciprocal of the c function. To avoid issues with division by zero, we take the maximum of c and 10^{-3}.

$$S^{t^1}_{r_1} = \sum_{\substack{t^2 \in T \\ t^1_{phr} \neq t^2_{phr} \\ W^{t^1}_{t^2} \neq 0}} \frac{1}{\max\left(c(r_1, t^2), 10^{-3}\right)} \tag{2.7}$$

$$S^{t^1}_{r_1} = \sum_{\substack{t^2 \in T \\ t^1_{phr} \neq t^2_{phr}}} \frac{W^{t^1}_{t^2}}{\max\left(c(r_1, t^2), 10^{-3}\right)} \tag{2.8}$$

Next we attempt some normalization of Equation 2.8 to ensure that scores are comparable to each other. This scoring function is called **Weighted Normalized Inverse**, given by Equation 2.9.

$$S_{r_1}^{t^1} = \frac{\sum_{\substack{t^2 \in T \\ t_{phr}^1 \neq t_{phr}^2}} W_{t^2}^{t^1} \left(\frac{\min_{r_1' \in R^{t^1}} \max\left(c(r_1', t^2), 10^{-3}\right)}{\max(c(r_1, t^2), 10^{-3})} \right)}{\sum_{\substack{t^2 \in T \\ t_{phr}^1 \neq t_{phr}^2}} W_{t^2}^{t^1}} \tag{2.9}$$

Equation 2.9 was constructed by multiplying the numerator in Equation 2.8 by the minimum distance between any result for t^1 and any result for t^2, not just between result r_1 and any result for t^2. Finally, we divide the whole expression by the total weight of all terms we looked at. This means that the score S_r^t is always between 0 and 1. A score of $S_r^t = 1$ means that when we consider the closest pair of results between t and another term, r is always the result from t that is part of that closest pair.

In scoring functions 2.7, 2.8, and 2.9 we skip all terms $t^2 \in T$ that have the same phrase as t^1. However, if a phrase appears many times in the text then it should be more important. So it might be helpful to give a bonus to the scores based on how often the phrase occurs. We add this feature to Equations 2.7, 2.8, and 2.9 to produce scoring functions **Inverse Frequency** (Equation 2.10), **Weighted Inverse Frequency** (Equation 2.11) and **Weighted Normalized Inverse Frequency** (Equation 2.12) respectively. In Equations 2.11 and 2.12 this is done by multiplying the original expression by the total weight of all terms with the same phrase. Since scoring function 2.7 is the same as 2.8 but with all non-zero weights set to 1, we similarly set all non-zero weights to 1 in Equation 2.10.

$$S_{r_1}^{t^1} = \left(\sum_{\substack{t^2 \in T \\ t_{phr}^1 \neq t_{phr}^2 \\ W_{t^2}^{t^1} \neq 0}} \frac{1}{\max\left(c(r_1, t^2), 10^{-3}\right)} \right) \sum_{\substack{t^2 \in T \\ t_{phr}^1 = t_{phr}^2}} 1 \tag{2.10}$$

$$S_{r_1}^{t^1} = \left(\sum_{\substack{t^2 \in T \\ t_{phr}^1 \neq t_{phr}^2}} \frac{W_{t^2}^{t^1}}{\max\left(c(r_1, t^2), 10^{-3}\right)} \right) \sum_{\substack{t^2 \in T \\ t_{phr}^1 = t_{phr}^2}} W_{t^2}^{t^1} \tag{2.11}$$

$$S_{r_1}^{t^1} = \frac{\sum\limits_{\substack{t^2 \in T \\ t_{phr}^1 \neq t_{phr}^2}} W_{t^2}^{t^1} \left(\frac{\min_{r_1' \in R^{t^1}} c(r_1', t^2)}{\max(c(r_1, t^2), 10^{-3})} \right)}{\sum\limits_{\substack{t^2 \in T \\ t_p^1 \neq t_p^2}} W_{t^2}^{t^1}} \sum\limits_{\substack{t^2 \in T \\ t_{phr}^1 = t_{phr}^2}} W_{t^2}^{t^1} \tag{2.12}$$

We have now created eight unique scoring functions, which we will test in Chapter 3. Combined with the two disambiguation algorithms discussed in the next section, this gives 16 versions of our algorithm to compare.

2.5.4 Disambiguation Algorithms

There are two versions of the disambiguation step of our location tagging algorithm, which are both explained here. The first version, called the 1-phase disambiguation algorithm, is described in Algorithm 3.

Algorithm 3 1-Phase Disambiguation Algorithm

1: Calculate $W_{t_2}^{t_1} \, \forall t_1, t_2 \in T$ using Equations 2.2, 2.3, and 2.4 (as described in Section 2.5.2)
2: **while** $(\exists p \in P | (|R^p| > 1))$ or $(\exists t_1, t_2 \in T | (W_{t_2}^{t_1} \neq 1))$ **do**
3:      **for all** $t \in T$ **do**
4:          **for all** $r \in R^t$ **do**
5:              Calculate S_r^t using scoring function 2.5, 2.6, 2.7, 2.8, 2.9, 2.10, 2.11, or 2.12
6:          **end for**
7:      **end for**
8:      $t^*, r^* \leftarrow (t \in T, r \in R^t)$ that maximize S_r^t and satisfy $(|R^t| > 1)$ or $(\exists t' \in T | W_t^{t'} \neq 1)$
9:      $R^{t_p^*} \leftarrow \{r^*\}$
10:     **for all** $t \in T$ **do**
11:         **if** $W_t^{t^*} = 0$ **then**
12:            $T \leftarrow (T \setminus \{t\})$
13:         **end if**
14:     **end for**
15:     Update P to reflect changes in T
16:     Recalculate $W_{t_2}^{t_1} \, \forall t_1, t_2 \in T$ as in step 1
17: **end while**

Step 2 repeats while there are still terms that need to be disambiguated. A term needs to be disambiguated if its phrase has more than one result, or if it conflicts with another term.

Step 8 is the key step in this algorithm. It finds the term t^* and result r^* that have the best score S_{r*}^{t*}. (In the case of a tie, the importance that Nominatim assigned to the results is used as the tie-breaker.) Step 8 only considers terms that need to be disambiguated.

After step 8 finds the result r^* with the best score, step 9 makes r^* the only result that is considered for phrase t_p^*. Also, steps 10 to 14 remove any terms that conflict with t^*. This means that after step 14 the term t^* has been completely disambiguated, and the algorithm is one step closer to disambiguating all terms in the text.

The other version of the disambiguation algorithm is called the 2-phase disambiguation algorithm, which is described in Algorithm 4. The difference between the 1-phase and 2-phase versions is that the 2-phase algorithm attempts to resolve all conflicting terms before disambiguating between search results.

Instead of one while-loop as we had in Algorithm 3, we have two. The first loop from steps 2 to 16 reduces the set T until there are no more conflicting terms. The second loop from steps 17 to 25 picks a search result for each remaining term. The reason for having these two phases is to resolve term conflicts as quickly as possible. These conflicts can have a large effect on result scores, especially with the unweighted scoring functions. If this effect is removed quickly, then it may improve the accuracy of the algorithm.

2.5.5 Example

Here we continue our example from Section 2.4.1. When we left that example we had just found search results from Nominatim for each phrase, shown in Table 2.3.

First we calculate the weights for each term. The weights are given in Table 2.7. Note that we have two interpretations for the "Georgian college" substring, and each has one interpretation. (Note that *Georgian* was tagged as an adjective, so it does not appear in its own term.)

Next we calculate the scores for each term. We will use the Weighted Distance scoring function, which was defined by Equation 2.6. We will also use the 1-Phase disambiguation algorithm (Algorithm 3).

Table 2.8 demonstrates how the scores are calculated. There is a row and a column for each result from Nominatim. Each cell contains the contribution to the score of the

---
Algorithm 4 2-Phase Disambiguation Algorithm

1: Calculate $W_{t_2}^{t_1}$ $\forall t_1, t_2 \in T$ using Equations 2.2, 2.3, and 2.4 (as described in Section 2.5.2)
2: **while** $\exists t_1, t_2 \in T | (W_{t_2}^{t_1} \neq 1)$ **do**
3:     **for all** $t \in T$ **do**
4:         **for all** $r \in R^t$ **do**
5:             Calculate S_r^t using scoring function 2.5, 2.6, 2.7, 2.8, 2.9, 2.10, 2.11, or 2.12
6:         **end for**
7:     **end for**
8:     $t^*, r^* \leftarrow (t \in T, r \in R^t)$ that maximize S_r^t and satisfy $\exists t' \in T | W_t^{t'} \neq 1$
9:     **for all** $t \in T$ **do**
10:         **if** $W_t^{t^*} = 0$ **then**
11:             $T \leftarrow (T \setminus \{t\})$
12:         **end if**
13:     **end for**
14:     Update P to reflect changes in T
15:     Recalculate $W_{t_2}^{t_1}$ $\forall t_1, t_2 \in T$ as in step 1
16: **end while**
17: **while** $\exists p \in P | (|R^p| > 1)$ **do**
18:     **for all** $t \in T$ **do**
19:         **for all** $r \in R^t$ **do**
20:             Calculate S_r^t using scoring function 2.5, 2.6, 2.7, 2.8, 2.9, 2.10, 2.11, or 2.12
21:         **end for**
22:     **end for**
23:     $t^*, r^* \leftarrow (t \in T, r \in R^t)$ that maximize S_r^t and satisfy $|R^t| > 1$
24:     $R^{t_p^*} \leftarrow \{r^*\}$
25: **end while**

Table cells contain $W_{t_2}^{t_1}$		t_1		
		RVH	$Georgian\ college$	$college$
t_2	RVH	1	1	1
	$Georgian\ college$	0.5	1	0
	$college$	0.5	0	1

Table 2.7: Example Weights

		RVH	Georgian college		college		
		1	1	2	1	2	3
RVH	1	N/A	1 * 0.7	1 * 42.7	1 * 86.7	1 * 4637.8	1 * 13166.5
Georgian college	1	0.5 * 0.7	N/A	N/A	N/A	N/A	N/A
	2	~~0.5 * 42.7~~	N/A	N/A	N/A	N/A	N/A
college	1	0.5 * 86.7	N/A	N/A	N/A	N/A	N/A
	2	~~0.5 * 4637.8~~	N/A	N/A	N/A	N/A	N/A
	3	~~0.5 * 13166.5~~	N/A	N/A	N/A	N/A	N/A
Total Score		-43.7	-0.7	-42.7	-86.7	-4637.8	-13166.5

Table 2.8: Example Score Calculation

column result from the row result. We do not consider scores between results for the same term, nor between results with a weight of 0, so those cells have been given N/A.

Since we are using the Weighted Distance scoring function, the score is the negative of the total distance between a result and the closest result for each other term. So for *RVH*, the closest result for *college* is about 86.7 km away. Farther results have been crossed out since they will not contribute to the score. The weight $W_{college}^{RVH}$ is 0.5, as we see in Table 2.7. Thus the contribution to the total score for *RVH* from *college* is about $0.5 * 86.7 = 43.35$. Similarly, the contribution from *Georgian college* is $0.5 * 0.7 = 0.35$, giving a final score of $-(43.35 + 0.35) = -43.7$ for result 1 for *RVH*.

Following Algorithm 3, we find the result and term with the greatest score. This is result 1 for *Georgian college*. We select this result to be a correct location reference in the text by removing all other results for *Georgian college*, and also removing any terms that conflict with *Georgian college*. This means removing *college* from T.

Now we recalculate the weights for our new set T. No terms conflict any more, which means that all weights are 1. Also, we observe that all remaining terms have exactly one result remaining. This means our disambiguation is complete, and we say that *RVH* refers to the Royal Victoria Regional Health Centre in Barrie, while *Georgian college* refers to the college in Barrie.

2.6   Summary

Now that each step in Algorithm 2 has been explained in more detail, we can write a more comprehensive summary of the entire algorithm. This is given by Algorithm 5.

Algorithm 5 Summary of Tagging Algorithm

1: Tag the input text using the POS and NER taggers from Stanford.
2: $S \leftarrow$ ordered list of words in the text
3: $T \leftarrow \emptyset$
4: **for** every $s \subseteq S$ where s is an ordered sequence of adjacent words **do**
5:     **if** s contains at least one word tagged as a LOCATION or noun **then**
6:         **if** every word in s is tagged as a noun, adjective, number, or LOCATION **then**
7:             $t \leftarrow$ a term made from the words in s
8:             $T \leftarrow (T \cup t)$
9:         **end if**
10:     **end if**
11: **end for**
12: **if** T contains terms that have words tagged with LOCATION **then**
13:     Remove terms from T that do not contain any words tagged with LOCATION.
14: **else**
15:     **if** T contains terms that occured after prepositions **then**
16:         Remove terms from T that did not occur after prepositions.
17:     **end if**
18: **end if**
19: **for** every $s \in S$ where s matches a Canadian, American, or Dutch postcode format **do**
20:     **if** s is not represented by any term in T **then**
21:         $t \leftarrow$ a term made from the words in s
22:         $T \leftarrow (T \cup t)$
23:     **end if**
24: **end for**
25: $P \leftarrow$ set of phrases represented in T
26: **for** every $p \in P$ **do**
27:     $R^p \leftarrow$ search results from the knowledge base for query p
28:     **if** $|R^p| = 0$ **then**
29:         **for** every $t \in T$ **do**
30:             **if** $t_{phr} = p$ **then**
31:                 $T \leftarrow (T \setminus \{t\})$
32:             **end if**
33:         **end for**
34:     **end if**
35: **end for**
36: Complete Algorithm 3 for 1-phase or Algorithm 4 for 2-phase disambiguation
37: **Return** set of non-conflicting terms T and one result in R^p for every $p \in P$.

At the end of Algorithm 5 we have a collection of terms that are mentioned in the text along with a single location for each. Scores can be recalculated for these results using the same scoring function that was used for the disambiguation. (Note that all weights will be 1 at this point.) Then they can then be sorted in order of decreasing score to provide a ranking for the top places mentioned in the text. This was done in Chapter 3 to compare the best results to the true location for articles.

Chapter 3

Results

In this chapter we describe the performance of our algorithm with each of the eight scoring functions that were presented in Section 2.5.3, along with both disambiguation algorithms that were discussed in Section 2.5.4.

3.1 Test Data

We tested our location tagging algorithm with a subset of English geotagged Wikipedia articles, that is, articles where the authors have provided latitude and longitude coordinates. This data required a significant amount of preprocessing, which is described in Appendix B.

Our full dataset contained 920,176 geotagged Wikipedia articles[1], but the tests in this chapter used only a sample of 5,976 articles. This subset was used due to time and resource constraints (in particular, the one query per second limit on the free Nominatim servers which was discussed in Section 2.4).

---

[1] It is worth noting that our set of articles is different than the one created by Wing and Baldridge [9], which was also used by Roller et al. [5]. They used a dump of the English Wikipedia from September 4, 2010 while we used a dump from June 2, 2015. Their dataset contained 10,355,226 articles (including non-geotagged ones), while ours contained 4,855,711. It should be expected that the more recent data would contain more articles, but the opposite is observed. However, Wing and Baldridge had to explicitly remove redirect articles, giving a total of 3,431,722 content-bearing articles. For this project we did not explicitly remove such articles, and none have been observed in our dataset. Therefore it is likely that these were automatically excluded in our download or by the processing software we used. Wing and Baldridge used their own processing software to extract article coordinates and obtained 488,269 geotagged articles. We used the set of coordinates that are directly available from Wikipedia.

Terms Used	Number of Articles	Percent of Articles
LOCATION tags only	5446	91.13%
Terms after prepositions	28	0.47%
All Terms	2	0.03%
Error	500	8.37%

Table 3.1: Filter Methods Used For Test Articles

3.2 Filtering by NER and POS Tags

Section 2.3.3 described how we filter our set of terms. If the text contains LOCATION tags, then we only use terms that have at least one word with this tag. If there are no LOCATION tags then we only keep terms that occur after prepositions. If there are no terms that follow prepositions, then we do not filter our set of terms.

Table 3.1 shows how often each of the filtering methods were used. Over 91% of articles were able to use terms with LOCATION tags. If this holds in general for other forms of text, then this implies that by restructuring this step in the algorithm we might improve its efficiency. Rather than adding all terms then discarding a large subset of them, we could instead add only the terms with LOCATION tags. Only when no LOCATION tags are found would the algorithm add other types of terms.

Table 3.1 also shows that over 8% of articles incurred an error. In all cases this was due to the implementation of our location tagging algorithm, which assumes that the POS tagger and NER tokenize the text in identical ways. If the implementation detects that there is not a one-to-one correspondence between how the two programs divided the text into words, then it stops and returns an error. This occurred in all 500 articles in the Error row of Table 3.1. Future work will be to determine why these differences occur and correct them so that the algorithm can continue for these articles.

3.3 Comparison of Disambiguation Algorithms

Our next step was to determine which versions of the algorithm are the most accurate. There are eight scoring functions and two disambiguation algorithms, giving a total of 16 versions to compare.

Each geotagged Wikipedia article in our data set has one set of coordinates which is considered to be the true location for the article. Our algorithm provided a list of terms

with one Nominatim result each. This means we needed to compare our list of multiple locations to the one true location for the article. We first measured error by calculating the distance between the true location and the result with the highest score, where scores were calculated using the same scoring function that the disambiguation algorithm used. (In Section 3.5 we will use a different error measurement.)

Articles were sorted in order of increasing error for each version of the algorithm. The fraction of these articles (out of 5476) were plotted along the horizontal axis of Figure 3.1, giving us the percentile of articles for which the disambiguation step was attempted.

Note that the horizontal axis of Figure 3.1 does not reach 100%. This is because disambiguation algorithms were terminated when their execution time exceeded 100 seconds. Out of 87,616 disambiguation attempts (16 algorithm versions times 5476 articles), this cutoff was used 496 times (0.57%).

The vertical axis gives the error in the geolocation. So, for example, this graph tells us that all versions of the algorithm had an error of less than one kilometre for at least 11% of articles.

We expected this graph to show one or two algorithms that were clearly better than the others, but this is not the case. All algorithms showed very similar results, and there was no clear winner. So instead we sliced the graph at the 10th, 25th and 50th percentiles, which are shown in Figures 3.2, 3.3, and 3.4 respectively.

Again there was no algorithm which was a clear winner. However, the 1-phase algorithm with the Weighted Inverse Frequency scoring function (Equation 2.11) was one of the two best algorithms in each of the three figures. Thus we chose it as our winning algorithm, and we will study it further in Section 3.4.

3.4   Further Analysis with the Winning Algorithm

There was no disambiguation algorithm that was clearly superior in our experiment (described in Section 3.3), but the Weighted Inverse Frequency scoring function (Equation 2.11) with the 1 phase disambiguation algorithm (Algorithm 3) seemed to be a good choice. In this section we further analyze the performance of the overall location tagging algorithm using this scoring function and disambiguation algorithm.
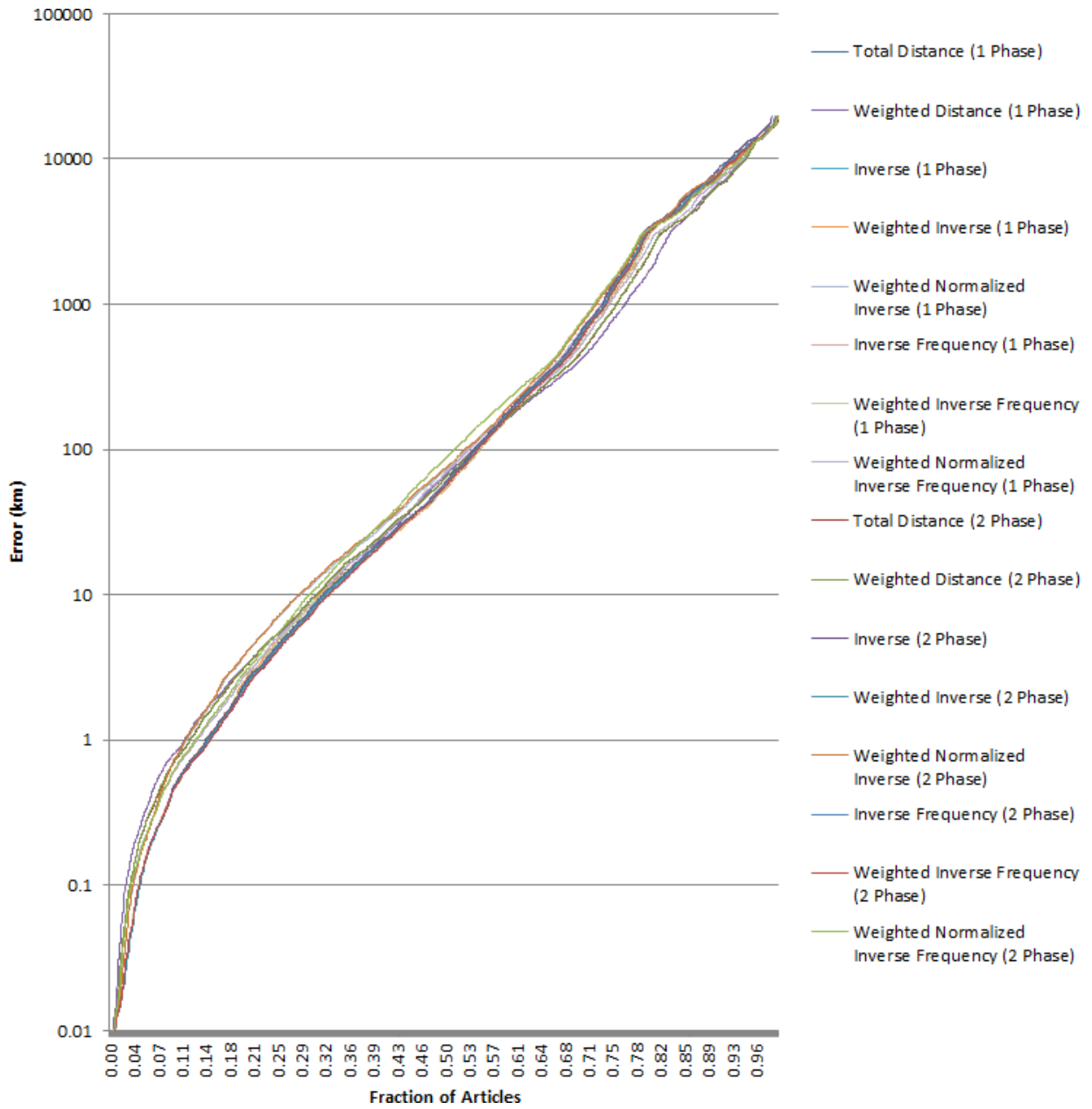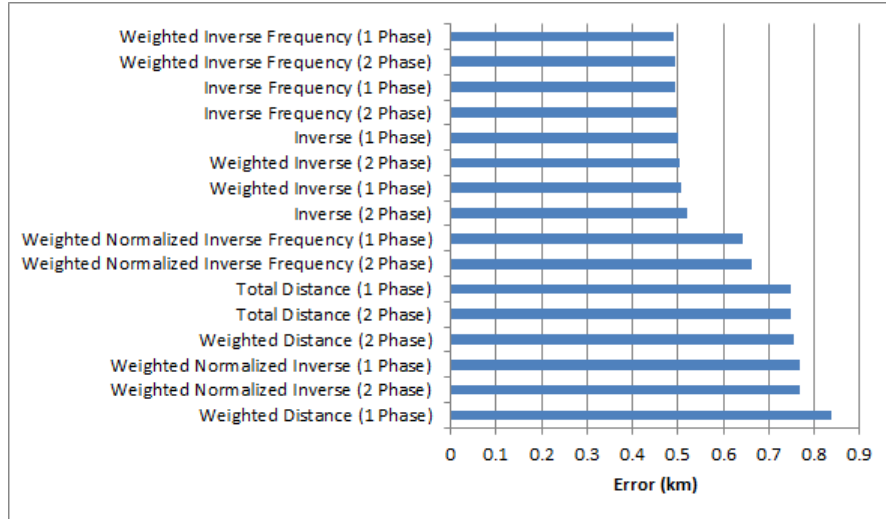
Figure 3.1: Error for Disambiguation Algorithms

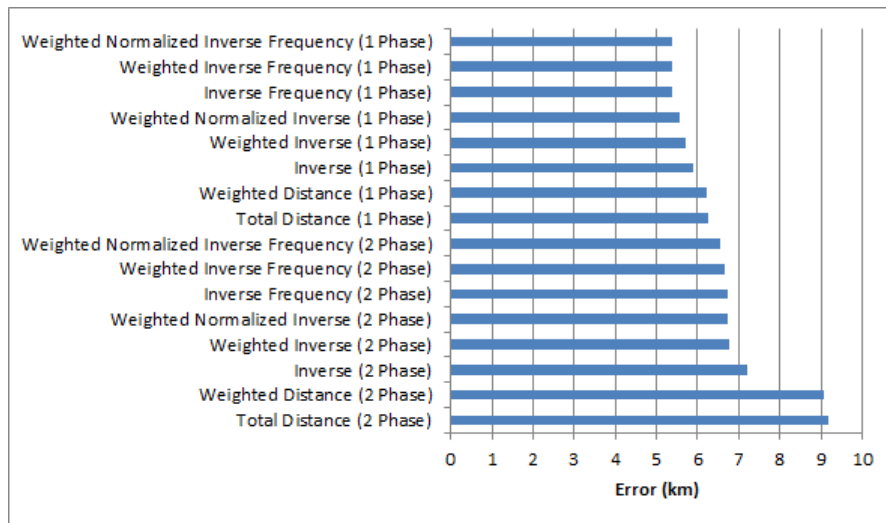Figure 3.2: Error for Disambiguation Algorithms at the 10th Percentile

Figure 3.3: Error for Disambiguation Algorithms at the 25th Percentile
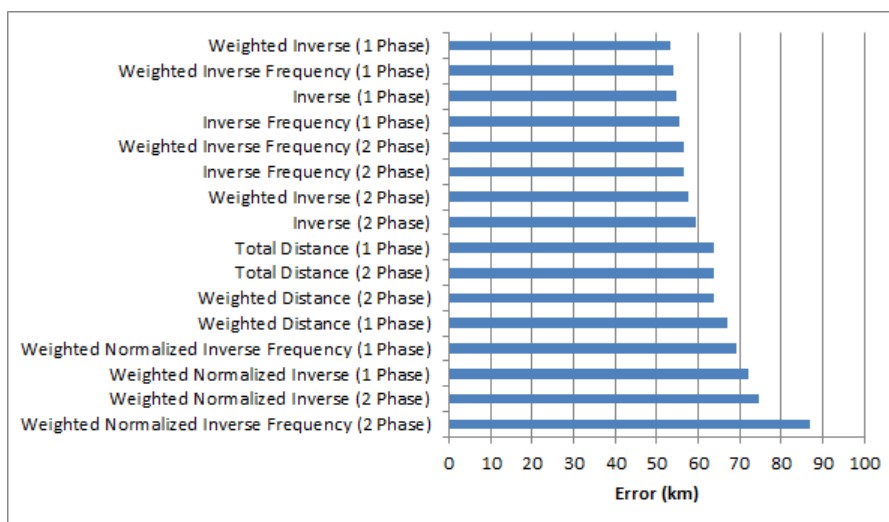
33

Figure 3.4: Error for Disambiguation Algorithms at the 50th Percentile

3.4.1 Results by Article Type

The author-provided location tags for Wikipedia articles each have a type associated with them. We investigated whether some of these types are more difficult for the algorithm to geolocate than others.

Nineteen different types of tags were observed in the test set, in addition to a NULL (or missing) type. Ten of these types, including the NULL type, occurred at least 50 times in the 5476 articles that were tested. Table 3.2 shows the median error (50th percentile) for each of these types.

We can see from Table 3.2 that airports and railway stations appear to be the easiest to geolocate, as they have the lowest median error. These types of articles are likely to name both the location and the nearby cities they serve, so our algorithm can expect to find a small cluster of results with good scores resulting in easy disambiguation.

Rivers and second-level administrative regions (districts, counties, etc.) appear to be the most difficult to locate. This makes sense, as these types of locations can be spread across a large area but are only represented by a single point in our algorithm. For example, if an article is about a river and our algorithm returns a location beside the river, this could show very poor accuracy if the returned point is far from the centre point that is assigned to the river by OpenStreetMap. The future work to correct this will be discussed in Section 4.1.3.

34

Article Type	Number of Articles	Median Error (km)
airport	60	6
railwaystation	178	6
waterbody	76	42
city	2414	44
mountain	124	54
landmark	1109	65
NULL	970	94
edu	187	110
river	86	187
adm2nd	105	295

Table 3.2: Accuracy Comparison for Different Article Types

3.4.2 Confidence Estimation

A valuable addition to our location tagging algorithm would be an estimator for the accuracy of the algorithm when the true error is unknown. This would allow us to assign a confidence value to each location tag.

Figure 3.5 shows the error in the location tagging as a function of the score of top result. It was expected that there would be a clear downward trend, where larger scores correlate to smaller errors. This would give us a simple relationship where the score of the top result would also be our confidence measure. However, our results did not indicate such a relationship and, as shown in Figure 3.5, there were three main clusters instead.

The bottom left cluster does show some of the desired trend, where larger scores indicate lower error. However, this is not simple to interpret, as there is another cluster directly above with a horizontal shape.

The third cluster is on the right, and spans the entire range of the graph. This cluster shows some vertical stratification starting around 1000 km. This is likely an artefact of the maximum we imposed on the denominator of Equation 2.11, where any distances less than 1 metre were increased to 1 metre as an attempt to avoid division by zero. Every time this restriction is imposed the score increases by 1000. This is likely the major cause of the stratification that is observed around the integer multiples of 1000 in Figure 3.5.

The results indicated in Figure 3.5 do not give us a simple answer in our search for a confidence measure. A few other confidence measures were tried with similar results, including the ratio between the top two scores, as well as the score multiplied or divided
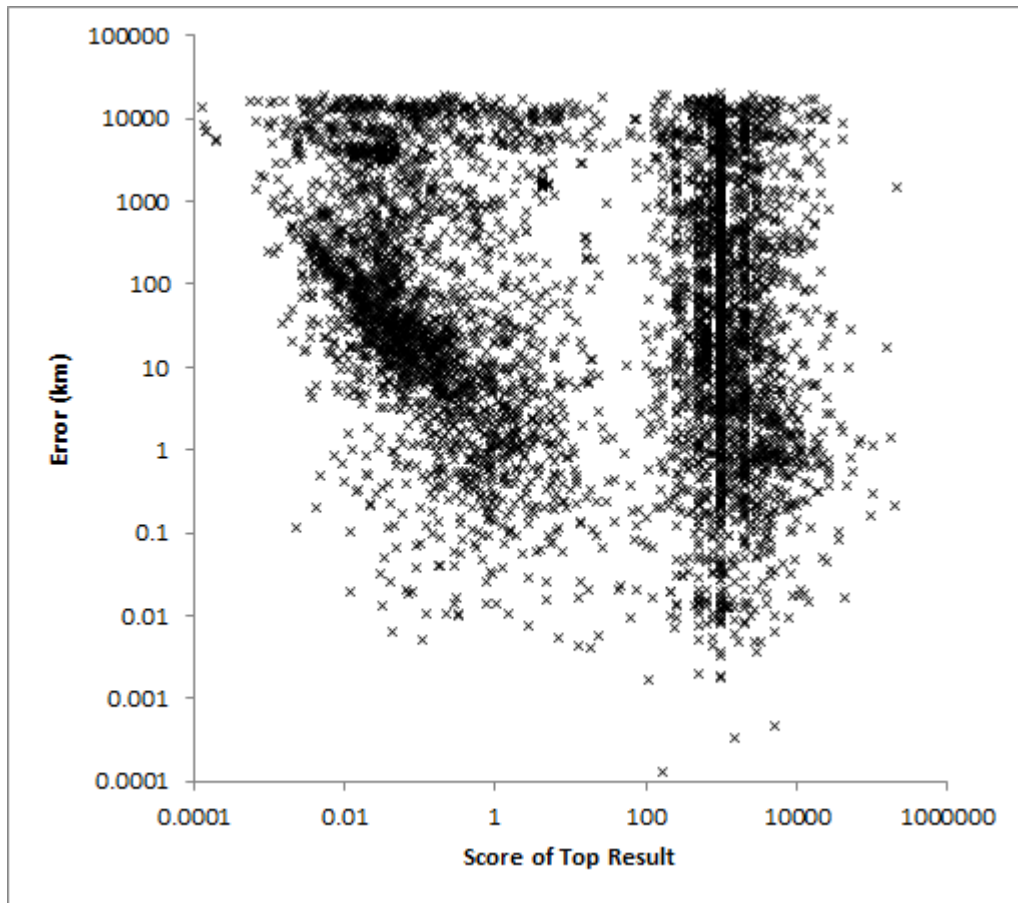
Figure 3.5: Error versus Scores for Weighted Inverse Frequency (1 Phase)

by the number of terms used to calculate the score. None of these showed a clearer relationship, so we do not yet have a confidence measure for our algorithm. This is left for future work.

We replotted the error in location tagging as a function of the score of the top result using the Weighted Distance (1-phase) version of the algorithm, and the result did not show the stratification that was apparent in Figure 3.5. However, there was still no clear relationship between score and error. Future work includes investigating all 16 versions, although given the similarities between them it is unlikely that this will result in a clear confidence measure.

3.5 Analysis of Top 5 Results

In previous sections we used the distance between the true location and the result with the greatest score as a measurement of error. This may not be the best approach, as our algorithm provides a list of multiple locations mentioned in the text. So far we used only one of these locations to calculate the error.

For example, if there is a text that describes a river and the cities that lie along it, then the location tagging algorithm should produce a location for each city as well as a location for the river. The true location for the article would be a single point somewhere along the river. If one of the cities on the river has the highest score, and this city is far from the point for the river, then this would give a very high error value even though our output may be considered correct.

It may make more sense to consider a group of locations that are returned by the location tagging algorithm. We re-defined our error measure to be the shortest distance between the true location and any of the five locations with the greatest scores, rather than just the top location with the greatest score.

Figure 3.6 is an updated version of Figure 3.1 which uses this Top 5 error measurement. Here we see the curves converge closer to each other than in Figure 3.1, meaning the 16 algorithms are more similar in this case. We also see a significant improvement in error. For example, the Weighted Inverse Frequency (1 phase) algorithm had a 10th percentile error of 490 metres with the Top 1 error, while for the Top 5 this error is 163 metres. Similarly, the error at the 25th percentile improved from 4.70 km to 0.87 km. Finally, the median error improved from 54 km to 8.5 km.

This new error measurement makes our errors similar to those observed by Roller et al. [5]. However, this error measure has access to the true answer can and pick the top 5

result that is closest to it. Despite this lack of experimental validity, it does tell us that we might significantly improve our accuracy by using a more-informed scoring function to rank the final results. This is left for future work.
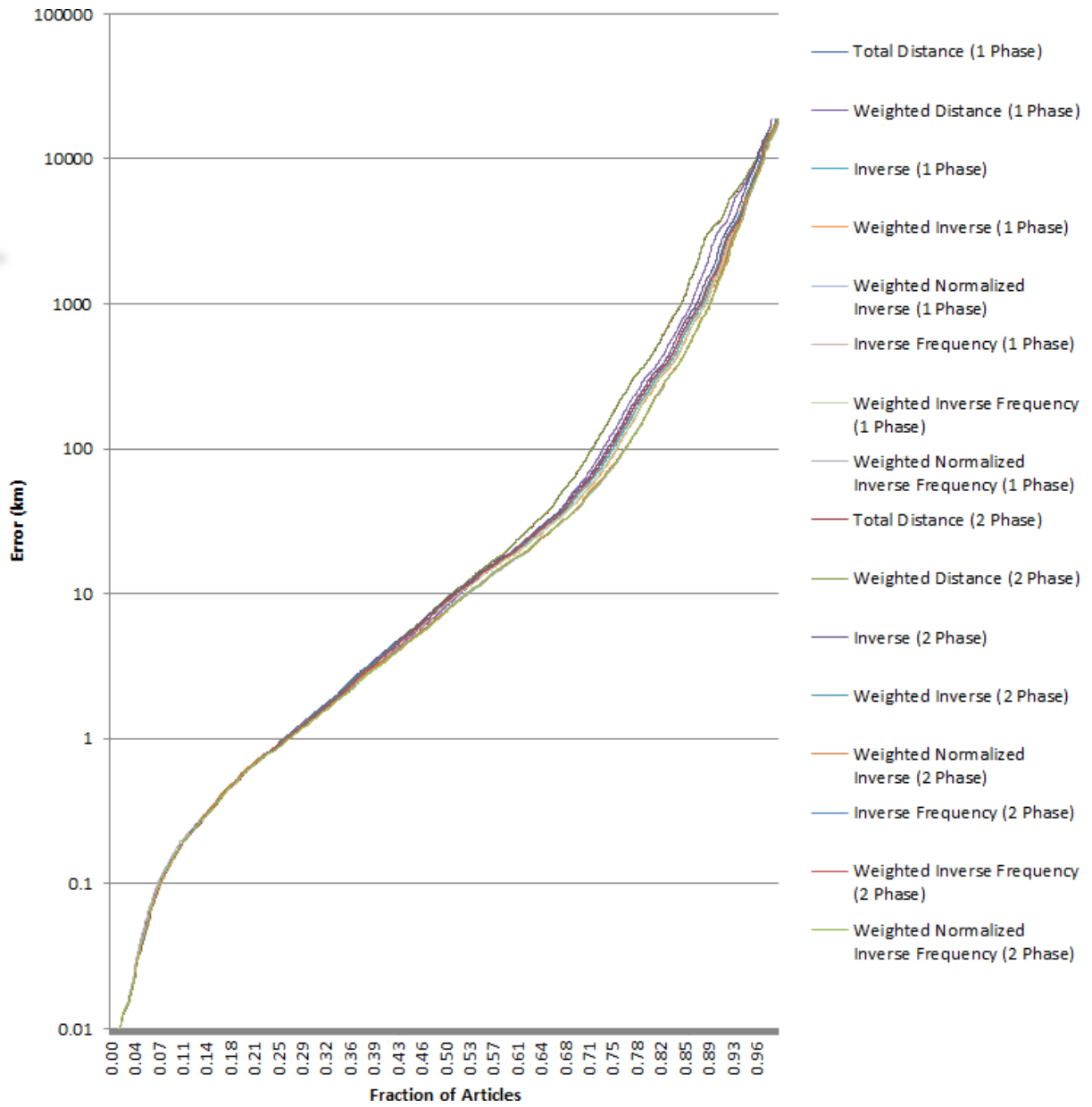
Figure 3.6: Top 5 Error for Disambiguation Algorithms

Chapter 4

Conclusion

This paper has presented an algorithm for finding precise locations that are named in text. It uses part-of-speech tagging and named entity recognition to find potential location references, which we call terms. A phrase is the text associated with a term. Our algorithm then uses the open source tool called Nominatim to search the OpenStreetMap database and obtain a list of possible locations for each unique phrase. Finally, conflicting terms are resolved and one location is chosen for each phrase. The algorithm returns a list of non-conflicting terms and the location that was chosen for each one.

Sixteen versions of the algorithm were compared, and none proved to be significantly better than the others. The Weighted Inverse Frequency scoring function (Equation 2.11) with the 1-phase disambiguation algorithm (Algorithm 3) was chosen as the best one, so we used this as our final version of the algorithm and reported accuracy using this version.

When the location with the highest score was chosen to represent the overall location of the article, our median error was 54 km. This is worse than the 11.0 km reported by Roller et al. [5]. However, when we consider the closest of the five highest-scoring results this median error reduces to 8.5 km. This implies that our algorithm, which is designed to return multiple locations, includes some very accurate results in most cases. Future work on the final sorting of results may significantly improve our 54 km median error.

4.1   Future Work

There are many ways in which our location tagging algorithm can be improved. Although the algorithm was developed heuristically, this was a good starting point for a new class

of algorithms for high-precision location tagging. Some improvements were discussed in previous chapters, and several more are described here.

4.1.1 Extraction and Disambiguation Feedback

Habib demonstrated that there is a feedback loop between the extraction step and the disambiguation step in named entity recognition [3]. Results from the disambiguation step can be used to improve results from the extraction step, which can then improve results for the disambiguation step. This concept was partially used in this project. In particular, the extraction step gathered all possible interpretations, then the disambiguation step refined the list of extracted terms by resolving those that conflicted.

Our location tagging algorithm could benefit from more use of this feedback loop. The disambiguation algorithm could be modified to discard terms which have extremely low scores, not just those that conflict with better terms, as these terms may not refer to real locations and should not have been included in the extraction step. This may improve accuracy when disambiguating the other terms in the text.

4.1.2 POS and NER Taggers

If this algorithm is to be applied to a specific type of text, then the algorithm could be improved by training part-of-speech taggers and named entity recognizers that are specifically trained for the target style of text. For example, if this algorithm were to be applied to tweets then the Stanford POS tagger we used could be replaced by the state-of-the-art Twitter POS tagger developed by Derczynski et al. [1].

4.1.3 Representation of Locations

As we described in Section 3.4.1, some extended areas can be difficult to geolocate, likely due to the representation of these areas in our algorithm. For example, consider the text "Waterloo is a city in Ontario." Nominatim represents each result for *Ontario* with a single point. This point can be hundreds of kilometres away from the city of Waterloo. Our location tagging algorithm should consider the distance between Waterloo and Ontario to be zero, as Ontario is the province which contains Waterloo. This change may significantly improve the accuracy of our algorithm, particularly for the large regions.

4.1.4 Postal Codes

As described in Section 2.3, the location extraction step uses regular expressions to search for postal codes in the text. It currently only searches for postal codes that follow the American, Canadian, and Dutch formats. This could be expanded to follow the formats of other countries.

Additionally, not all postal codes can be found in OpenStreetMap. Some postal codes are sent to Nominatim and receive no results. If OpenStreetMap were updated to contain all postal codes, then texts which contain those postal codes can expect to receive more accurate results.

APPENDICES

Appendix A

External Software

The following software was used in the implementation of this project. All links were at the time of writing and are subject to change.

- Apache Spark version 1.4.0. Based on the work of [10]. Available at http://spark.apache.org/.

- Nominatim. It was queried using the API described at http://wiki.openstreetmap.org/wiki/Nominatim. It can also be used through a user interface at https://nominatim.openstreetmap.org/.

- PHP-Stanford-NLP. October 18, 2014 version. Written by Anthony Gentile. Available at https://github.com/agentile/PHP-Stanford-NLP. Used as a PHP-wrapper to access the Stanford POS tagger and NER.

- Stanford Log-linear Part-Of-Speech Tagger version 3.5.2 (April 20, 2015). Based on the work of [8] and [7]. Available at http://nlp.stanford.edu/software/tagger.shtml. There are multiple pre-trained models available with this software, and this project used the model titled *english-left3words-distsim*.

- Stanford Named Entity Recognizer version 3.5.2 (April 20, 2015). Based on the work of [2]. Available at http://nlp.stanford.edu/software/CRF-NER.shtml. There are multiple pre-trained models available with this software, and this project used the model titled *english.all.3class.distsim.crf*.

- WikiExtractor. June 14, 2015 version. Written by Giuseppe Attardi. Available at https://github.com/attardi/wikiextractor.

Appendix B

Wikipedia Data

At the time of writing, data dumps of the English Wikipedia could be obtained from https://dumps.wikimedia.org/enwiki/. This report used articles from a dump of the English Wikipedia on June 2, 2015 (file name enwiki-20150602-pages-articles.xml.bz2). Location tags were also obtained from the June 2, 2015 dump, and are available as a separate download on the same site (file name enwiki-20150602-geo_tags.sql.gz).

The raw dump file was processed using WikiExtractor (https://github.com/attardi/wikiextractor), which takes the XML file and produces the raw text for each article, removing special formatting and annotations such as images, tables, or references. However, the output of WikiExtractor does not contain location tags, so we had to join this data with the set of location tags ourselves.

The geo_tags file contains all location tags in the English Wikipedia. There are two types of tags: primary and non-primary. Primary tags apply to an entire article. Non-primary tags apply to a particular mention in the text of an article. This project only used primary tags.

These two data sets were joined using Apache Spark, which is a scalable cluster computing system [10]. Spark now has a highly active community of both developers and users. Spark's APIs made it easy to read in both data sets, match location tags to their articles, and output in a clear format using less than 80 lines of Scala code. The output of this Spark program was directly usable by our implementation of our location tagging algorithm.

References

[1] Leon Derczynski, Alan Ritter, Sam Clark, and Kalina Bontcheva. Twitter part-of-speech tagging for all: Overcoming sparse and noisy data. In *Recent Advances in Natural Language Processing*, pages 198–206, 2013.

[2] Jenny Rose Finkel, Trond Grenager, and Christopher 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)*, pages 363–370, Stroudsburg, PA, USA, 2005.

[3] Mena Badieh Habib Morgan. *Named entity extraction and disambiguation for informal text: the missing link*. PhD thesis, Enschede, May 2014. SIKS Dissertation Series No. 2014-20.

[4] Bo Han, Paul Cook, and Timothy Baldwin. Text-based twitter user geolocation prediction. *Journal of Artificial Intelligence Research*, 49(1):451–500, January 2014.

[5] Stephen Roller, Michael Speriosu, Sarat Rallapalli, Benjamin Wing, and Jason Baldridge. Supervised text-based geolocation using language models on an adaptive grid. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 1500–1510, Stroudsburg, PA, USA, 2012.

[6] Beatrice Santorini. Part-of-speech tagging guidelines for the Penn Treebank project (3rd revision). Technical report, Department of Linguistics, University of Pennsylvania, Philadelphia, PA, USA, 1990.

[7] Kristina Toutanova, Dan Klein, Christopher D. Manning, and Yoram Singer. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational*

Linguistics on Human Language Technology (HLT-NAACL), pages 173–180, Strouds-burg, PA, USA, 2003.

[8] Kristina Toutanova and Christopher D. Manning. Enriching the knowledge sources used in a maximum entropy part-of-speech tagger. In *Proceedings of the 2000 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing (EMNLP) and Very Large Corpora (VLC): Held in Conjunction with the 38th Annual Meeting of the Association for Computational Linguistics*, pages 63–70, Stroudsburg, PA, USA, 2000.

[9] Benjamin P. Wing and Jason Baldridge. Simple supervised document geolocation with geodesic grids. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics (ACL): Human Language Technologies (HLT)*, pages 955–964, Stroudsburg, PA, USA, 2011.

[10] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Mur-phy McCauley, Michael J. Franklin, Scott Shenker, and Ion Stoica. Resilient dis-tributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Imple-mentation (NSDI)*, Berkeley, CA, USA, 2012.