WIKIMANTIC:

PROCESSING THE SEMANTIC MEANING OF TEXT

by

Christopher Boston

A thesis submitted to the Faculty of the University of Delaware in partial fulfillment of the requirements for the degree of Master of Science in Computer Science

Spring 2012

WIKIMANTIC:

PROCESSING THE SEMANTIC MEANING OF TEXT

by

Christopher Boston

Approved: _____

Sandra M. Carberry, Ph.D.
Professor in charge of thesis on behalf of the Advisory Committee

Approved: _____

Errol L. Lloyd, Ph.D.
Chair of the Department of Computer and Information Sciences

Approved: _____

Babatunde A. Ogunnaike, Ph.D.
Dean of the College of Engineering

Approved: _____

Charles G. Riordan, Ph.D.
Vice Provost for Graduate and Professional Education

ACKNOWLEDGMENTS

This work uses Microsoft Web N-gram Services and was supported by the National Science Foundation under Grants III-1016916 and IIS-1017026.

I would like to thank Sandra Carberry, my advisor, for her limitless patience and impeccable advice. The hours she spent reviewing my drafts and listening to my half-formed theories over the past year were crucial to my arrival at a final paper of which I am genuinely proud. I would also like to thank Hui Fang, who provided invaluable input on both the theory and implementation of Wikimantic. Together, Sandra and Hui were vital for transforming a simple idea about query expansion into a full-fledged thesis on disambiguation with a strong theoretical backing and working implementation.

I would like to thank my family for all the support they have given me and all the things they have taught me. Dad, who showed me how to think analytically and find wonder in the way the world works. Mom, who taught me how to work with words and communicate my thoughts to others. Kevin, whose positive attitude and competitive spirit has taught me the value of challenging myself and learning from failures. They have shaped me in so many ways that this thesis is as much a reflection of them as it is me, and their contribution runs even deeper than the considerable support they have provided during my time at UD.

TABLE OF CONTENTS

LIST OF TABLES . . . . . **vi**
LIST OF FIGURES . . . . . **vii**
ABSTRACT . . . . . **viii**

Chapter

1 INTRODUCTION **1**

1.1 Wikimantic's Role in Information Graphic Retrieval 2
1.2 Problem Definition 3

2 RELATED WORK **5**

2.1 Previous Solutions 5
2.2 Wikimantic's Contribution 7

3 APPROACH **9**

3.1 Generative Model 9

3.1.1 Concepts 10
3.1.2 Atomic Concepts 11
3.1.3 Mixture Concepts 12

3.2 Interpreting Text with a Topic Concept 14

3.2.1 Populating the Topic Concept 15
3.2.2 Weighting the Topic Concept according to the Generative Model 15
3.2.3 ReferenceRank 16

3.3 Disambiguation 18

3.3.1 Determining the Number of References 18

3.3.2 Product Method 19
3.3.3 Mixture Method 19
3.3.4 Alternative Methods 20

4 EVALUATION 22

4.1 Experiment Setup 22
4.2 Results 23

5 CONCLUSION AND FUTURE WORK 26

5.1 Conclusion 26
5.2 Future Work 27

5.2.1 Conditional probabilities 28
5.2.2 Practical applications of conditional probability calculations . 29

BIBLIOGRAPHY 32

LIST OF TABLES

4.1 Performance on Nouns Only 23

4.2 Performance on all Non-function Words 23

LIST OF FIGURES

3.1 Product method's scoring of four possible disambiguations of "new york city" 19

3.2 Mixture method's scoring of four possible disambiguations of "new york city" 20

3.3 Tagger method's scoring of three possible disambiguations of "new york city" 21

ABSTRACT

This thesis presents an implemented and evaluated methodology for disambiguating terms in search queries. By exploiting Wikipedia articles and their reference relations, our method is able to disambiguate terms in particularly short queries with few context words. This work is part of a larger project to retrieve information graphics in response to user queries.

Chapter 1

INTRODUCTION

Disambiguation is the fundamental, yet tantalizingly difficult problem of annotating text so that each ambiguous term is linked to some unambiguous representation of its sense. Wikipedia, the online encyclopedia hosted by the Wikimedia Foundation, has garnered a lot of interest as a tool for facilitating disambiguation by providing a semantic web of hyperlinks and "disambiguation pages" that associate ambiguous terms with unambiguous articles [1, 2, 3, 4]. For an encyclopedia, English Wikipedia is monolithic. It contains over 3.5 million articles which are interconnected by hundreds of millions of user-generated links. Although errors do exist in articles and link structure, Wikipedia's strong editing community does a good job of keeping them to a minimum.[1]

Wikipedia has several features that make it an excellent resource for disambiguation systems. First, Wikipedia articles are organized so that each one has a single focused topic. All text that appears in an article can be safely assumed to discuss that article's topic. Second, editors are encouraged to augment important words in their articles with links so that a reader can click a word and read an article that discusses it. Augmented words are called "anchor text", and the links that augment them provide clues as to which topics the anchor text could refer. Third, Wikipedia provides disambiguation pages that list all articles that could be referenced by a given

[1] Here, we refer primarily to technical issues such as duplicate articles or dead links. However, a common criticism of Wikipedia is that its open nature makes the article content susceptible to bias and error. For our purposes, we actually prefer common misconceptions and biases to appear in Wikipedia's text since they are likely to appear in our input text as well.

term. For example, there is a disambiguation page for the term "apple" that lists the article *Apple_Inc.*, *Apple_*$(fruit)$, *Apple_Records*, and more. Articles, links, and disambiguation pages are jointly created by many human editors, so they are sensitive to semantic meaning in a way that is difficult to find in other corpuses.

1.1 Wikimantic's Role in Information Graphic Retrieval

This document describes the methods and performance of Wikimantic, a system designed to disambiguate short queries. Wikimantic is part of a larger digital library project to retrieve information graphics (bar charts, line graphs, etc.) that appear in popular media such as magazines and newspapers. Such graphics typically have a high-level message that they are intended to convey, such as *Visa ranks first among credit cards in circulation.* Others[5, 6] have developed a system for identifying this high-level message. We anticipate retrieving graphics relevant to a user query by relating the query to a combination of the graphic's intended message, any text in the graphic, and the context of the associated article. To do this, we must first disambiguate the words in the query.

Although there are many existing methods that extract semantic information via disambiguation, most require large amounts of context terms or focus exclusively on named entities [3, 4, 7, 8]. Our experiments have shown that most queries are very short and that words other than named entities must be disambiguated. Thus, a more robust method of disambiguation is required.

Our work has several novel contributions to disambiguation which are important for information retrieval systems. First, we disambiguate text strings that to our knowledge are the shortest yet. Second, our method is robust with respect to the terms that can be disambiguated, rather than being limited to nouns or just named entities. And third, our method can determine when a sequence of words should be disambiguated as a single entity rather than as a sequence of individual disambiguations. Furthermore, our method does not rely on capitalization since users are notoriously poor at correct capitalization of terms in their queries; this is in contrast to the text

of formal documents where correct capitalization can be used to identify sequences of words that represent a named entity.

1.2 Problem Definition

When we disambiguate a term, we are trying to identify the concept to which the term *refers*. For the purpose of this paper, we say that a term refers to a concept when a speaker (or author) uses that term to communicate the concept to a listener (or reader). For example, the term "Apple" refers to $Apple\_Inc$ when an author writes "Apple" and means the company. $Apple\_Inc$ is said to be the target of that reference. Occasionally, the speaker will employ terms that do not directly refer to any concept. These terms have no correct target and should be ignored. In order to identify such terms, we make the simplifying assumption that all content words refer to a concept and all function words do not. We say a term is salient if and only if it refers to some concept.

When we disambiguate a sequence of terms, we seek to map each salient term to the Wikipedia article that best represents the term in context. More formally, let $s = (t_1, t_2, \ldots, t_{|s|})$ be a sequence of $|s|$ terms. For every term t_j, if t_j is salient (not a function word), we wish to generate a mapping $t_j \rightarrow C_i$ where C_i is the Wikipedia article that best defines the concept t_j referenced. For example, given the sentence "Steve Jobs resigns from Apple", an acceptable mapping would link "steve" and "jobs" to the Wikipedia article about Steve Jobs, the former CEO of Apple. "resigns" would be mapped to the article titled Resignation, and "apple" would be mapped to the article for Apple Inc. Mapping "apple" to the article Apple_(fruit) would be unacceptable in that context.

Thus, the output W of a disambiguation system can be seen as an ordered collection of mappings. The correct output of "Steve Jobs resigns from Apple" would be the following:

$$W = [(\text{“steve”} \rightarrow Steve\_Jobs), (\text{“jobs”} \rightarrow Steve\_Jobs), (\text{“resigns”} \rightarrow Resignation),$$

(“apple” $\rightarrow$ *Apple_Inc.*)]

It is important that “steve jobs” is broken into two mappings, even though they collectively refer to the same man. This ensures that the outputs of two disambiguation systems on the same input text will always contain the same number of mappings, which keeps scoring consistent. A system cannot “cheat” by consolidating incorrect mappings together or breaking a single correct mapping down into many correct mappings. Additionally, function words like “from” do not appear at all in the output. Such words have no correct mapping and may be stripped from the input text before the disambiguation system processes it.

The remainder of this document is organized as follows. Chapter 2 discusses related research and Wikimantic’s contribution to it. Chapter 3 describes the underlying principles and mechanics of Wikimantic’s method of disambiguation. In particular, section 3.1 explains the theory behind our method and introduces the Concept abstractions we use to represent semantic information. Sections 3.2 and 3.3 explain how the theory and Concept abstractions are exploited to interpret a text string and arrive at a coherent set of disambiguations. Chapter 4 presents an evaluation of the Wikimantic system and chapter 5 concludes with a general summary and suggestions for future work.

Chapter 2

RELATED WORK

2.1 Previous Solutions

Bunescu and Pasca are generally credited with being the first to use Wikipedia as a resource for disambiguation [7]. They formulated the disambiguation task to be a two step procedure where a system must (1) identify the salient terms in the text and (2) link them accurately. To identify salient terms, they constructed a dictionary of about 500,000 named entities from Wikipedia and performed lookups to see which terms could conceivably reference a known entity. However, there is a difference between discovering salient terms and identifying the number of references that those terms encode. "Romeo and Juliet" would appear in the dictionary as the name of the play by Shakespeare. However, the dictionary alone is not enough to decide whether "Romeo and Juliet" constitutes a single reference to the play *Romeo_and_Juliet* or two separate references to the lead characters *Romeo* and *Juliet*, who each have their own articles in Wikipedia. Although Bunescu and Pasca's method is capable of detecting salience, they seem to make the strong assumption that only one reference will ever exist in their input string.

While Bunescu and Pasca's work was initially limited to named entity disambiguation, Mihalcea[2] later developed a more general system that linked all "interesting" terms. Mihalcea's keyword extractor and disambiguator relied heavily on anchor text extracted from Wikipedia's inter-article links. When evaluating the disambiguator, Mihalcea gave it 85 random Wikipedia articles with the linked terms identified but the link data removed, and scored it based on its ability to guess the original target of each link. Mihalcea achieved good precision and recall on this task, and we believe

the vast amount of information in Wikipedia can be leveraged to perform even more difficult tasks. In our evaluations, we assume that we are given full sentences that contain both salient terms and unimportant terms that may not even be covered by Wikipedia. This differs significantly from Mihalcea's evaluation, where the only terms in the test set that needed to be annotated were those that were already proven suitable for annotation by the human Wikipedia editors. Where Mihalcea was concerned with imitating Wikipedia's sparse links, we seek to identify and link as many terms as possible.

Many Wikipedia based disambiguation systems use variants of Mihalcea's method which attempt to match terms in the text with anchor text from Wikipedia links [3, 9]. When a match is found, the term is annotated with a copy of the link. Sometimes, a term will match anchor text from multiple conflicting links, in which case the system must choose between them. Milne and Witten's[3] contribution was to look for terms that matched only non-conflicting links, and use those easy disambiguations to provide a better context for the more difficult ones. Given a large enough text string, it's always possible to find at least one trivial term to start the process. However, short strings do not reliably contain trivial terms.

Ferragina and Scaiella [1] addressed this problem by employing a voting system that resolved all ambiguous terms simultaneously. They found that good results were attainable with text fragments as short as 30 words each, which would allow for the disambiguation of brief snippets from search engine results or tweets. Although their results are very good, 30 words is still too large for the short queries we wish to process. In our evaluation, we limit our full sentence queries to a maximum of 15 terms in length. The average query length in our test set is just 8.9 words, including stop words.

Ratinov et. al [4] define a local disambiguation method to be one that disambiguates each term independently, and a global disambiguation method to be one that searches for the best set of coherent disambiguations. Their recent work has shown that the best ranking performance can usually be obtained by combining local and global approaches. Although their system was limited to named entities, our performance

seems to be best when combining our own local and global approaches as well.

2.2 Wikimantic's Contribution

Wikimantic makes three major contributions to the above line of work. First, we disambiguate text strings that to our knowledge are the shortest yet. This opens up a wide range of applications that were previously closed to disambiguation systems. Short text fragments like queries, captions, and labels on graphs can now be disambiguated on their own. Most of the previous systems [1, 2, 3, 4, 8] require a large amount of context which is often unavailable.

Second, our method is robust with respect to the terms that can be disambiguated. Wikimantic does not limit itself to just named entities, and it seeks to disambiguate terms that are not precisely covered in Wikipedia by finding the best available article. This means that more semantic information is extracted from fewer words, leading to a more precise model of the query.

Third, our method does not require references to be identified beforehand. In many disambiguation systems[2, 4, 7, 8], the input includes information about which terms constitute a reference to some unknown concept. For example, Ratinov's named entity disambiguator is built on the assumption that the document is passed to the system along with a set of "mentions" which identifies every substring in the document that must be disambiguated. The sentence "Steve Jobs resigns from Apple" would have the accompanying set ("Steve Jobs", "Apple"). This greatly simplifies disambiguation since there is no chance of mistakenly deciding that "Steve" and "Jobs" constitute two separate references to the name *Stephen* and *Job_(Employment)*. One may wonder whether it is realistic to assume that a preprocessing system could generate this set of mentions without disambiguating them in the first place.

These contributions make Wikimantic more viable for information retrieval, especially when using short queries to search a corpus of information graphics. Since graphics present their primary message visually, only small amounts of text can be scraped from labels and captions. Even when a graphic is accompanied by an article,

it is dangerous to rely entirely on the article text for disambiguation. During a corpus study[10] to determine to what extent a graphic's primary message was available in an accompanying document, it was found that over half of the documents captured little or none of the graphic's primary message. Wikimantic's ability to disambiguate more words with less context allows us to avoid relying too heavily on the accompanying article for context terms when disambiguating the graphic. It also allows us to accurately disambiguate the short queries which have no accompanying article.

Chapter 3

APPROACH

This chapter covers Wikimantic's approach to word sense disambiguation. We introduce the process with a high level summary and then explain each step in detail.

We begin by defining a generative model which serves as the theoretical underpinning for Wikimantic's implementation. The model is used to relate words and ideas in a way that is both conceptually intuitive and mathematically well defined. Although the model itself makes no direct claims as to the definitions and relationships of ideas and words, it does provide a framework for extracting this data from Wikipedia. Using the generative model in conjunction with Wikipedia, it is possible for Wikimantic to estimate the probability of a particular idea being relevant to a particular text string. With these estimates, Wikimantic can more or less bridge the gap between terms explicitly written in a text and the implicit semantic content of the text. Section 3.3 explains how we exploit this ability to perform word sense disambiguation.

3.1 Generative Model

Authors encode ideas into words and put the words on paper. A reader may later take these words and decode them back into ideas. Our generative model is based on the premise that every idea has certain associated words that are used to talk about the idea. A person writing about the Apple Corporation may use terms like "computer", "iPhone", "Steve", or "Jobs". A reader can use a priori knowledge about these term-concept associations to know that the writer means Apple Corporation and not the fruit when they just say "apple".

Our generative model makes the simplifying assumption that it is the ideas themselves that generate terms in a text. When a writer wishes to write a document or formulate a query about the Apple Corporation, we say that the idea of *Apple_Corporation* is actually generating the terms in the text directly. Therefore, a query about *Apple_Corporation* is likely to contain terms like "computer" and "iPhone" due to the idea *Apple_Corporation*'s propensity to generate such terms.

3.1.1 Concepts

We employ the object oriented paradigm to express the Generative Model in terms of precisely defined polymorphic objects. The ideas that generate words in the text are represented by *Concept* objects. All objects of type Concept are defined to have two important properties.

First, a Concept C has the property $P(C)$, the prior probability that any given term in any text will be generated by that Concept. For example, if *Europe* and *Queuing_theory* are Concepts, one would expect $P(Europe)$ to be relatively high and $P(Queuing\_theory)$ to be relatively low. Many documents and conversations involve *Europe*, but discussions about *Queuing_theory* are comparatively rare. Thus, $P(C)$ is a measure of how likely it is that an author will write about C , using C to generate terms in the text.

Second, a Concept C has the property $P(t|C)$ for every term t. $P(t|C)$ is the probability that term t will be generated given that Concept C is generating. This is just a formal way of saying that certain words are used to talk about certain Concepts. One would expect $P(\text{"}computer\text{"}|Apple\_Corporation)$ to be relatively high and $P(\text{"}chicken\text{"}|Apple\_Corporation)$ to be comparatively low.

Our generative model states that a text is generated term by term from some *topic Concept*. The a priori probability of a given Concept C being the topic Concept of our text is denoted $P(C)$. For every term t, there is a probability $P(t|C)$ that C will generate t as the next term in the text.

Documents, queries, and other forms of text are all considered to be of one type, TermSequence. If $P(C)$ and $P(t|C)$ are known, it is possible to take the terms in a TermSequence and work backwards to find the probability that C generated the TermSequence (see section 3.2.2). In order to get $P(C)$ and $P(t|C)$, we extract a set of fundamental AtomicConcepts from Wikipedia.

3.1.2 Atomic Concepts

An AtomicConcept is defined to be a special subtype of Concept for which $P(C)$ and $P(t|C)$ are known. By convention, we use A instead of C to represent AtomicConcepts, so $P(A)$ and $P(t|A)$ are used to denote the prior and conditional probabilities of a Concept which is Atomic.

We view each article in Wikipedia as a long TermSequence that was generated by some AtomicConcept. Since every article is unique, there is a one to one mapping between articles and the AtomicConcepts that generate them. Everything that is known about an AtomicConcept is approximated by examining the corresponding Wikipedia article.[1]

Wikipedia editors are encouraged to annotate their text with links to other Wikipedia articles that discuss AtomicConcepts mentioned in the annotated article. Because of this, AtomicConcepts that are discussed frequently tend to be linked often, and the fraction of incoming links can be used as a rough estimate of how often an AtomicConcept is discussed. Thus, $P(A)$ is estimated by counting the number of inter-article links in Wikipedia that point to A.

$$P(A) = \frac{number\ of\ incoming\ links}{number\ of\ links\ in\ Wikipedia}$$

[1] Since each AtomicConcept is entirely defined by its corresponding article, we often use the words "AtomicConcept" and "article" interchangeably. For example, if we say that AtomicConcept A links to AtomicConcept B, we are really saying that the corresponding article for AtomicConcept A links to the corresponding article for AtomicConcept B.

To estimate $P(t|A)$, we view the article body text as a sample of terms generated by A. The probability of A generating a term t is:

$$P(t|A) = \frac{count(t, A)}{number\ of\ words\ in\ A}$$

Because articles have finite length, some terms relevant to A won't actually show up in the body text of the article. For each term not present in the article, we smooth the distribution by estimating the probability of A generating t to be the probability of t occurring in the English language.[2]

Each AtomicConcept instance is a data structure that encodes important information about a single idea. The important information, like term frequencies and incoming links, can each be represented as Vectors. These Vectors can in turn be stored in a single Vector. Thus, an AtomicConcept is really just a Vector of Vectors, and therefore the operations of addition and scalar multiplication can be applied to AtomicConcepts. This property allows us to define a new kind of Concept called a MixtureConcept in the next section.

3.1.3 Mixture Concepts

A MixtureConcept is defined to be a special subtype of Concept for which $P(C)$ and $P(t|C)$ are unknown. For example, the topic Concept of the sentence "Steve Jobs resigns from Apple" will not exist in our set of known AtomicConcepts because Wikipedia does not contain an article specifically devoted to the resignation of Steve Jobs. Wikipedia does, however, contain articles that cover *Steve_Jobs*, *Apple_Inc*, and *Resignation* separately. To model an unknown topic, we construct a MixtureConcept by expressing it in terms of other Concepts which are already known. By convention,

[2] In Wikimantic, we use Microsoft n-Grams to give us P(t). Because probabilities from the article and Microsoft n-Grams each sum to 1, the sum of $P(t \mid A)$ over all possible terms equals 2. In practice, estimated probability values for AtomicConcepts are always stored as elements of normalized collections, which ensures that no probability value falls outside the range [0,1]

we use M instead of C to represent MixtureConcepts, so $P(M)$ and $P(t|M)$ are used to denote the prior and conditional probabilities of a Concept which is a mixture.

Formally, a MixtureConcept is defined to be a weighted Vector of n Concepts, where all the weights sum to 1. When a MixtureConcept generates a term, it randomly selects one of the Concepts in its Vector to generate in its stead. The weight of a Concept in the Vector tells us the probability that it will be the one selected to generate.

$$\text{Let MixtureConcept } M = \sum_{i=1}^{n} w_i * C_i$$
$$\text{where } w_i = \text{the weight of } C_i \text{ in } M$$

Thus, a MixtureConcept M can be viewed as a series of weighted Concepts added together. The Concepts can be AtomicConcepts or MixtureConcepts, but ultimately M is defined by AtomicConcepts since they are the fundamental building blocks we used to build the MixtureConcepts in the first place. We can exploit this fact to simplify MixtureConcepts as sums of AtomicConcepts even though they are technically allowed to contain Concepts of any type. We will use this to show that for any linear function over AtomicConcepts, one can always construct an equivalent function that operates on MixtureConcepts.

Let f be a linear function that takes an AtomicConcept A and returns some output y.

Let $M = \sum_{i=1}^{n} w_i * C_i$ be a MixtureConcept of n Concepts.

Due to the Additivity property of linear functions,

$$f(A_0 + A_1 + ... + A_n) = f(A_0) + f(A_1) + ... + f(A_n)$$

Due to the Homogeneity property of linear functions,

$$\alpha * f(A_0) = f(\alpha * A_0)$$

Since M can be written as the weighted sum of AtomicConcepts,

$$f(M) = f(w_0 * A_0 + w_1 * A_1 + ... + w_n * A_n) = w_0 * f(A_0) + w_1 * f(A_1) + ... + w_n * f(A_n)$$

Thus, any linear function over AtomicConcepts can also be applied to MixtureConcepts[3]. Since all Concepts are either AtomicConcepts or MixtureConcepts, such functions can always be applied to Concepts.

$$f(C) = \begin{cases} f(A), & \text{if } C \text{ is of type AtomicConcept} \\ f(M), & \text{if } C \text{ is of type MixtureConcept} \end{cases}$$

In section 3.1.2, we defined functions to compute $P(A)$ and $P(t|A)$. These functions can be applied to MixtureConcepts due to their Additive and Homogenous properties.

$$P(M) = \sum_{i=1}^{n} w_i * P(C_i) \tag{3.1}$$

$$P(t|M) = \sum_{i=1}^{n} w_i * P(t|C_i)$$

3.2 Interpreting Text with a Topic Concept

Let s be a TermSequence we wish to summarize. The generative model states that each TermSequence was generated term by term from some topic Concept. We wish to interpret s by building a MixtureConcept T which models its topic Concept. Constructing T is a two step procedure in which we populate with AtomicConcepts and then weight them. Our base method uses the content of an AtomicConcept's Wikipedia article to estimate its weight in T, but these weights can optionally be refined with an algorithm called ReferenceRank. Either way, the result is a weighted sum of AtomicConcepts which together constitute the MixtureConcept T.

---

[3] Interestingly, the reverse is also true. An AtomicConcept A can always be typecast to an equivalent MixtureConcept by defining M to be a Mixture of just A. $M = \sum_{i=1}^{n} 1 * A$. Thus, any linear function $f(M)$ over MixtureConcepts can be applied to any AtomicConcept.

3.2.1 Populating the Topic Concept

To find AtomicConcepts that belong in T, we look at every subsequence of terms in s and attempt a direct lookup in Wikipedia. Any article that has a title that matches a subsequence of terms in s is added to T. Any article that is disambiguated by a page whose title matches a subsequence of terms in s is also added to T. Finally, all articles that share a disambiguation page with an article already in T are added. For example, if s = "Steve Jobs resigns":

Steve → Matches title of Disambiguation page "Steve". Add all articles disambiguated by that page.

Jobs → Matches the title of a redirect page that points to Jobs_(Role). Add Jobs_(Role) and all other articles that Jobs_(disambiguation) link to.

Resigns → Matches the title of a redirect page that points to Resignation. Add Resignation and all other articles that Resignation_(disambiguation) link to.

Steve Jobs → Matches the title of article for Steve Jobs, the entrepreneur.

Jobs Resigns → Matches nothing, so no articles added.

Steve Jobs Resigns → Matches nothing, so no articles added.

3.2.2 Weighting the Topic Concept according to the Generative Model

Once our MixtureConcept T is populated, it will contain a large number of Concepts of varying degrees of relevance, and we rely on weighting to diminish the influence of spurious Concepts.

We weight each Concept according to the probability that every term in s was generated by that Concept in sequence, ignoring stopwords.

$$w_i = P(C_i|s) = \prod_{j=1}^{|s|} P(C_i|t_j) \tag{3.2}$$

$$P(C_i|t_j) = \frac{P(t_j|C_i) * P(C_i)}{P(t_j)}$$

This weighting schema ensures that a Concept will only get a high score if it is likely to generate all terms in the sequence. A Concept like $Jobs\_(Role)$ may have a high probability of generating "jobs", but its low probability of generating "steve" will penalize it significantly. We can expect the Concept $Steve\_Jobs$ to generate "steve", "jobs", and "resigns" relatively often, which would give it a larger weight than $Jobs\_(Role)$ would get. Occasionally there will be an unrelated Concept that happens to have a high probability of generating all context words. To avoid assigning undue significance to such spurious Concepts, the weights of T can be refined with an additional algorithm called ReferenceRank.

3.2.3 ReferenceRank

ReferenceRank is an algorithm that takes T as an input and refines the weights to produce a new topic Concept T_R. Consider the following examples where unrefined T might be misleading.

$$T_1 = 0.5 * Apple\_Inc. + 0.5 * Whole\_Foods$$
$$T_2 = 0.5 * Apple\_Inc. + 0.2 * iPhone + 0.2 * Apple\_Safari + 0.1 * iPad$$

In the text described by T_1, the topic is 50% about Apple Inc. and 50% about the grocery store Whole Foods. In the text described by T_2, the topic is 50% about Apple Inc. and 50% about various Apple products. Since $iPhone$, $Apple\_Safari$, and $iPad$ are all Concepts that are likely to generate the term "apple" (referring to the company), one would expect Apple Inc. to be referenced more often in T_2 than T_1. However, Apple Inc. is weighted equally in T_1 and T_2. It's subtle, but there is a very real difference between the probability that a Concept will generate terms in our query and the probability that a Concept will appear in our query. To account for this, we extend our generative model by making the claim that Concepts generate references to other Concepts as well as terms.

When AtomicConcept A_1 generates a reference, the probability that the referenced Concept is A_2 is estimated as the probability that clicking a random link in A_1's article will lead directly to the article for A_2.

$$P(R_{A_2}|A_1) = \frac{number\ of\ links\ from\ A_1\ to\ A_2}{total\ number\ of\ links\ originating\ at\ A_1}$$

Remember that in section 3.1.3 we showed that any function that takes AtomicConcepts can be applied to MixtureConcepts as well. Since all Concepts are either AtomicConcepts or MixtureConcepts, we can calcuate $P(R_{C_2}|C_1)$ for any Concepts C_1 and C_2. For example, if C_1 and C_2 were both MixtureConcepts $P(R_{C_2}|C_1)$ would be calculated as follows.

$$P(R_{C_2}|C_1) = \sum_{i=1}^{|C_1|} \sum_{j=1}^{|C_2|} w_i * w_j * P(R_{C_2[j]}|C_1[i])$$

where $C_1[i]$ is the i^{th} Concept in C_1

and $C_2[j]$ is the j^{th} Concept in C_2

To compute T_R, we populate it with every Concept from T and weight each Concept by T's probability of generating a reference to that Concept.

$$\text{Let topic Concept } T_R = \sum_{i=1}^{n} w_i * C_i$$

$$\text{where } w_i = P(R_{C_i}|T)$$

This process is very similar to one iteration of the PageRank[11] algorithm, where nodes in a graph vote for other nodes to which they link. In our case, Concepts in T vote for Concepts in T_R using the links in their Wikipedia articles. The power of a Concept's vote is proportional to the weight of that Concept in T.

In section 3.2.2, we described how to weight Concepts in T according to their probabilities of generating s. In this section, we explained how the weights of T can be refined to produce T_R, where Concepts are weighted according to the probability of being the target of a reference generated by T. Since the goal of disambiguation is to identify $term \rightarrow Concept$ references, one would expect T_R to provide all the information needed to identify the most probable Concepts. As it turns out, T_R's reliance on Wikipedia's relatively sparse link structure can make it unreliable at times.

"Do Life Savers cause tooth decay?" is a perfectly reasonable query, but there are no direct links between the articles for $Life\_Savers$ and $Tooth\_decay$. This means that neither will vote for the other and their weights in T_R will be much lower than one would expect. Although T_R is useful when links are found, it must be supplemented with information from T. For this reason, the Concept $T_M = (1 - d) * T + d * (T_R)$ is used. The optimal value of d is determined experimentally.

3.3 Disambiguation

In this section, we describe how we use our weighted Concept to arrive at a final collection of disambiguation mappings. Our general approach here is to generate a number of candidate collections and find the most likely one by scoring them with one of two methods.

3.3.1 Determining the Number of References

In many disambiguation papers[2, 4, 7, 8], the important term strings are assumed to be marked ahead of time and the system must simply choose the single best Wikipedia article for the marked string. In real world queries, the number of mappings are not known a priori, which makes disambiguation considerably more difficult. Does "life saver" refer to the brand of candy or a person who saved a life? If we are talking about junk food, then "life saver" should entail a single mapping to the AtomicConcept $Life\_Saver$, otherwise it entails two separate mappings to $Life$ and $Saver$. Although these kinds of conflicts seem like they should be rare, the vast coverage of Wikipedia actually makes them common. Company names, book titles, and music album titles are particularly troublesome since they are often common phrases; moreover, they are often the topics of graphs in popular media and thus occur in user queries for these graphs.

Disambiguation would be straightforward if these conflicts didn't occur. For each term t, one could simply choose the AtomicConcept A that maximizes $P(A|s)$ from the list of all AtomicConcepts that were added to T by t in Section 3.2.1.

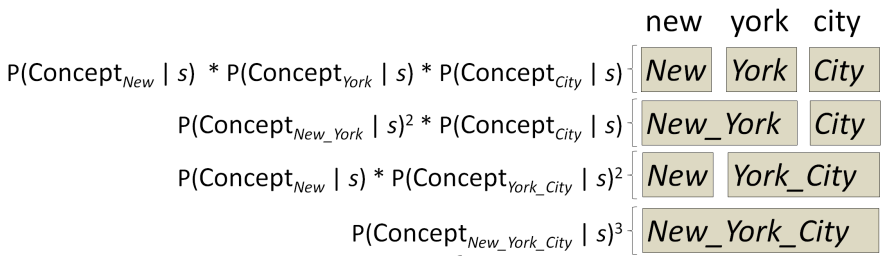

Figure 3.1: Product method's scoring of four possible disambiguations of "new york city".

However, if a problematic sequence of terms like "life saver" or "new york city" is found, every possible breakdown of the sequence must be considered separately. Each breakdown yields a unique candidate collection of disambiguations that our approach scores according to its probability of being the correct one. The scoring is calculated using either the Mixture method or the Product method.

3.3.2 Product Method

The Product method scores a candidate collection as the product of the probabilities of each mapping of term to AtomicConcept. Figure 3.1 depicts the four candidate collections that are considered when the string "new york city" is broken down. We use italics to refer to AtomicConcepts by name, so $P(Concept_{New}|s)$ refers to the probability of the Concept New being the disambiguation of the term "new". The first collection contains the three AtomicConcepts New, $York$, and $City$. The score of the collection is simply the product of their probabilities multiplied together. When n adjacent terms should be disambiguated as a single entity, the Product method scores it as n disambiguations of the entity, as shown by the fourth row in Figure 3.1, where the score for the sequence "new york city" is $P(Concept_{New\_York\_City})$ to the third power.

3.3.3 Mixture Method

The Mixture method treats a collection of possible disambiguations as a mixture of the AtomicConcepts that disambiguate terms in the collection. The AtomicConcepts in $M_{collection}$ are given equal weight. Under the Mixture method, a collection's score

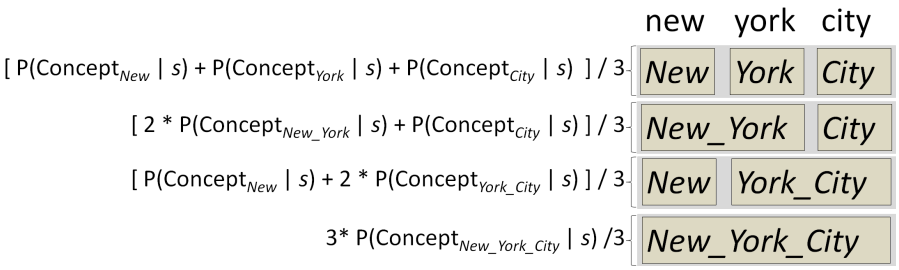

Figure 3.2: Mixture method's scoring of four possible disambiguations of "new york city"

is simply equal to the average of the probability values of all AtomicConcepts in the collection; once again, n adjacent terms that were disambiguated as referring to a single entity are counted as n disambiguations. For example, the fourth line of Figure 3.2 shows the sequence "new york city" being disambiguated as a single entity but the score in this case is just the probability of the concept *New_York_City* (ie., the average of the scores for three disambiguations of the sequence).

3.3.4 Alternative Methods

When formulating the Product and Mixture methods, we devised a number of alternative methods for choosing the best collection of possible disambiguations. We ruled out these methods during our preliminary tests, but feel that it may be illuminating to describe them here.

The "Use-Best" method was a simple alternative where the system would resolve conflicts by choosing the collection that contained the single most probable Concept, no matter how unlikely the other Concepts in the collection may be. The intuition was that it was more important to rely on our most certain Concept than worry about the unlikely mappings, but the method had issues when the input text contained multiple conflicts. If *Steve_Jobs* is the most likely concept in "steve jobs apple computer", any collection that resolves "steve jobs" correctly will get the maximum score, regardless of how well it resolves "apple computer". Use-Best was not capable of achieving good performance due to its simplistic scoring.

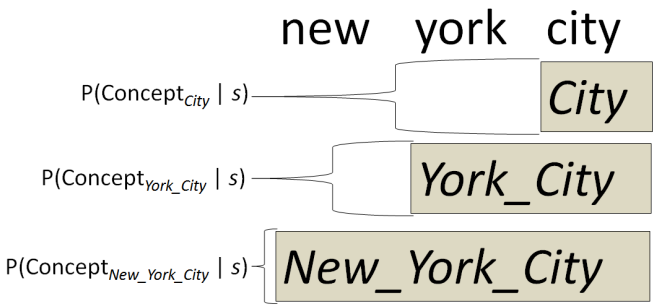

Figure 3.3: Tagger method's scoring of three possible disambiguations of "new york city"

The Tagger method was an alternative that first analyzed the text with a part of speech tagger and broke the sentence into noun phrases. For each noun phrase, the Tagger method considers every possible substring of the phrase that ends with the last term in that phrase. Each substring is assumed to be a collective reference to a single concept, and the single most likely reference in the noun phrase is selected as correct. The strength of the Tagger method is that it greatly simplifies scoring. There is never a situation in which a set of multiple mappings must be compared against a single mapping. Additionally, there are much fewer possible disambiguations to compare, which makes things more efficient. Unfortunately, there are some critical disadvantages that make it infeasible from a practical and theoretical standpoint. First, the method's accuracy is greatly limited by accuracy of the part of speech tagger. A single incorrect tag can break a noun phrase apart or join two of them together, causing every term in the phrase to be incorrectly disambiguated. Since the performance of this method is intrinsically linked to the performance of the part of speech tagger, it is difficult to isolate the weaknesses of the method from the weaknesses of the tagger during evaluation. Second, the tagger expects to receive grammatically correct full sentences as input. This greatly limits the potential uses of Wikimantic, and seems to be an unreasonable constraint for a system that operates on actual user queries which may contain grammatical errors.

Chapter 4

EVALUATION

4.1 Experiment Setup

Wikimantic includes four possible combinations of methods for disambiguation: the Product and Mixture methods with T as the topic Concept and the Product and Mixture methods with (1-d)*T+d*T_R as the topic Concept. Each method was evaluated using 70 queries from the Trec 2007 QA track and 26 queries collected for our Information Graphic Retrieval project. Since the goal of Wikimantic was to handle short grammatically correct full sentence questions, we found that the QA track suited our needs fairly well. The queries acquired from the Information Graphic Retrieval Project were collected from human subjects who were given information graphics and told to write queries they might have used to find them.

All queries contain at least one (but usually more) salient word that must be disambiguated. The word count of each query is no less than 4 and no greater than 15. Out of the 850 words in the set, evaluators identified about 349 nouns (they disagreed on a couple due to ambiguous phrasing of the queries). About 110 words were content words that were not nouns. We present results for disambiguating just nouns and for disambiguating all non-function words.

To measure correctness, we gave the system output to two human evaluators and instructed them to decide for each term whether the mapped page correctly described the meaning of the word as it was used in the query. The general rule was that a mapping was wrong if a significantly better page could be found for the term. For non-nouns, it was considered correct if a verb or adjective was linked to its noun-equivalent article. For example, it would be acceptable to annotate the term “defect” (to betray)

with the page *Defection*. If a term appeared in the query with a sense that has no equivalent article in Wikipedia, the evaluators were instructed to mercilessly mark the output wrong.

4.2 Results

Tables 4.1 and 4.2 present statistics on precision and recall for the four methods. Precision is equal to the number of terms correctly mapped to concepts divided by the number of terms mapped by the system. Recall is equal to the number of correct mappings divided by the number of terms fed to the system. All Precision and Recall measures are displayed as percentages.

Performance (Nouns Only)				
	Topic Concept: T		Concept: (1-d)*T+d*T_R	
	Mixture	Product	Mixture	Product
Precision	78.71	80.51	81.38	82.76
Recall	75.21	76.93	77.65	79.08
F-Measure	76.92	78.68	79.47	80.88

Table 4.1: Performance on Nouns Only

Overall, the Product method fared better than the Mixture method, and performance was better on nouns than on non-nouns. With the Mixture method, it's possible for an obviously incorrect mapping of one term to be offset by a high scoring

Performance (All Terms)				
	Topic Concept: T		Concept: (1-d)*T+d*T_R	
	Mixture	Product	Mixture	Product
Precision	66.82	68.28	69.52	70.57
Recall	61.47	62.45	63.96	64.61
F-Measure	64.04	65.23	66.62	67.46

Table 4.2: Performance on all Non-function Words

mapping of another. With the Product method, a mapping with a near-zero probability will cause the score for the entire collection to be near-zero. The Product method is therefore a more conservative scoring method that favors well rounded collections over collections with some very likely and very unlikely references.

Wikimantic's exceptional performance on nouns seems to be partly due to Wikipedia's greater coverage of nouns. Certain verbs like "established", "occur", and "withdraw" don't have any article in Wikipedia, even though they are relatively common terms. Verbs also have many conjugations which can lead to errors since Wikimantic does not support stemming. A detailed analysis of Wikimantic's output revealed that precision on non-nouns was 25%, significantly worse than precision on nouns (82.76%).

Fortunately, most salient terms seem to be nouns. In our test queries, we found that 349 out of 459 content words were nouns. Interestingly, the ratio of nouns to non-nouns differed greatly between the QA test set and the Information Graphic Retrieval Project queries. The QA test set contained 253 nouns and only 62 non-nouns, which is a 4:1 ratio. The Information Graphic Retrieval Project queries contained 97 nouns and 46 non-nouns, a 2:1 ratio. Although the combined test set contains mostly nouns, a closer examination seems to suggest that not all text is written the same way. To get optimal results with Wikimantic, it is important to be aware of the ratio of nouns to non-nouns in the text that is being processed.

For each of the two methods described in Section 3.3, we evaluated Wikimantic using the mixture $(1-d)*T+d*T_R$ with varying values of d. Improved performance occurred for small values of d (d<.2). Although the optimal value of d was found to be very small (d = 0.0001), the effects of ReferenceRank were still surprisingly significant. MixtureConcepts often get weighted in such a way that one AtomicConcept has virtually all the weight, which gives it extremely high voting power. The top AtomicConcept's votes are then so powerful that they have disproportionate sway over the lesser AtomicConcepts. The small value of d works to balance this out.

Based on our results, the approach of interpreting text with Concepts extracted

from Wikipedia seems to work. The best results are achieved when mixing T with T_R and then applying the Product method to arrive at the best collection of disambiguation mappings. Even with short queries, we are able to get good precision and recall values that suggest Wikimantic has real potential as a disambiguator.

Chapter 5

CONCLUSION AND FUTURE WORK

5.1 Conclusion

In this paper, we formulated a theoretical model which served as the basis of the Wikimantic implementation. Using this theoretical model, we are able to provide an intuitive and mathematical backing for our system that both illuminates and validates the most fundamental operations of Wikimantic. Our model is also extensible. In section 3.1.3, we showed that any linear function over AtomicConcepts can also be applied to MixtureConcepts in a meaningful way. Thus, the theoretical model supports the addition of new functions without altering the fundamental framework. Although Wikimantic was implemented and evaluated using the September 2011 dump of English Wikipedia, our model assumes very little about the actual content of the underlying wiki. In theory, the model could be used with other language versions of Wikipedia to apply the principles of Wikimantic to other languages.

We implemented and evaluated Wikimantic to demonstrate the feasibility of using the theoretical model on actual search queries. We found that although our implementation worked most effectively on nouns, it was robust enough to handle queries that included other parts of speech as well. Our method is unique in that it can handle exceptionally short queries without reliance on capitalization or prior knowledge about which terms collectively refer to single entities. Employing the ReferenceRank algorithm helped refine the results so that resulting disambiguations were more coherent. Overall, we are satisfied with the performance of Wikimantic when it comes to word sense disambiguation, and confident in Wikimantic's potential as the subject of future work.

5.2 Future Work

For the purpose of this paper, disambiguation is a good task to demonstrate Wikimantic's adaptability and sensitivity to context when interpreting a string. However, there are more features of Wikimantic that have a lot of potential for use in an actual IR system.

In section 3.2, we explained how to convert a TermSequence to a topic Concept T that encodes a summary of the TermSequence's implicit semantic content. NLP systems are often concerned with relevance between texts, and one may wonder if there is a meaningful way to measure the relevance between any two arbitrary Concepts. Up to this point, we assumed any system that incorporated Wikimantic would compute relevance by disambiguating the texts and comparing the results. In practice, relying on disambiguation to guide relevance judgments will cause two serious issues that may effect accuracy.

First, errors incurred during disambiguation will be silently propagated through to the relevance calculation. Suppose that a disambiguation system is run on two strings that mention $Apple\_Inc$, but the term "apple" from one of the strings is erroneously disambiguated to $Apple\_(fruit)$. If one were to naively compare the disambiguations of the two strings, they would appear to discuss very different Concepts and receive a relevance score that is too low.

Second, two relevant texts could be correctly disambiguated but share no references to the same concepts. One would expect a text about bicycles to be tangentially related to a text about motorcycles, even if the text about motorcycles never explicitly mentions bicycles. However, it would be difficult for a system that relies entirely on the output of a disambiguator to detect the relevance of the two related yet distinct Concepts.

The source of both of these issues seems to be in the definition of the disambiguation task. To conform to the task requirements, Wikimantic must take each term and respond with the single most likely Concept to which the term refers. If Wikimantic identifies other less likely (but still plausible) alternatives, they will not be included

in the output. In this way, disambiguation necessarily leads to information loss, especially when the disambiguator is forced to make poorly informed guesses about truly ambiguous text.

Fortunately, this information loss only occurs at the very end of Wikimantic's disambiguation process, after the topic Concept is estimated but before the final disambiguation candidate is selected. To compare the relevance of two texts, it might be more effective to compare the topic Concepts of each text directly.

5.2.1 Conditional probabilities

In section 3.2.3, we employed ReferenceRank to refine a Concept's weight according to its relevance with all other Concepts in the topic Concept. ReferenceRank relied on the assumption that two relevant Concepts would link to each other, but was somewhat unreliable due to the sparsity of linked words in article text. This prompted us to formulate a new method which examines more links and has a stronger mathematical backing. Instead of unreliably trying to divine an estimate for the subjective notion of relevance, we instead compute the conditional probability $P(C_1 \mid C_2)$ of discussing a Concept C_1 given that another Concept C_2 is known to be discussed. As usual, we will first define this function in terms of AtomicConcepts, and defer to the results of section 3.1.3 to guarantee that it must apply to all Concepts.

Suppose we wish to find the coreference probability of AtomicConcept A_1 given AtomicConcept A_2. By the definition of conditional probability:

$$P(A_1 \mid A_2) = \frac{P(A_1 \cap A_2)}{P(A_2)}$$

If we stick to our assumption (from section 3.1.2) that a link to an AtomicConcept implies it was discussed, we can interpret the above formula to get the following:

$$P(A_1 \mid A_2) = \frac{number\ of\ articles\ that\ link\ to\ both\ A_1\ and\ A_2}{number\ of\ articles\ that\ link\ to\ A_2}$$

Essentially, we define the coreference probability of AtomicConcept A_1 given AtomicConcept A_2 to be the conditional probability that some article that links to A_2 will link to A_1 as well. For example, to compute $P(Philosophy \mid Socrates)$, we would divide the number of articles that simultaneously link to *Philosophy* and *Socrates* by the total number of articles that link to *Socrates*. Since nearly every discussion about Socrates is also about philosophy to some extent, we would expect $P(Philosophy \mid Socrates)$ to be very high. Interestingly, coreference probabilities are *not* necessarily symmetric. Socrates is just one of many philosophers, and it is entirely possible to discuss *Philosophy* without ever mentioning *Socrates*. Therefore, one might expect $P(Socrates \mid Philosophy)$ to be much lower than $P(Philosophy \mid Socrates)$, though still larger than something irrelevant like $P(Socrates \mid Hard\_disk\_drive)$. [1]

5.2.2 Practical applications of conditional probability calculations

This method of calculating conditional probabilities has many applications, and not just as a replacement for ReferenceRank. When combined with our method of generating a topic Concept, we have an extremely powerful tool for directly comparing text strings. Suppose we wish to compare string s_1 ="steve jobs announces iphone" with string s_2 ="apple ipad sale". Using the topic Concept estimation method from section 3.2, we can construct topic Concepts C_1 and C_2 for the strings. We can then compute $P(C_1 \mid C_2)$ or $P(C_2 \mid C_1)$ to find the probability that the semantic content of string s_1 will appear in a discussion including the semantic content of string s_2.

Comparing strings in this way has three very important advantages over traditional string comparisons that operate on terms directly. First, Wikimantic is able to recognize strings that discuss similar topics even when the strings themselves have no terms in common. In the above example, Wikimantic can pretty confidently generate precise topic Concepts for s_1 and s_2 because they contain the terms "iphone"

[1] Although this method has not yet been formally evaluated on a large test set as of this writing, our current version of Wikimantic does indeed yield the expected results for this particular example.

and "ipad", which frequently appear in the articles for *Steve_Jobs* and *Apple_Inc*. *Steve_Jobs*, *Apple_Inc*, *iPhone*, and *iPad* cooccur frequently in Wikipedia, meaning many of their incoming links originate from the same articles. This leads to a high value for $P(C_1 \mid C_2)$ and $P(C_2 \mid C_1)$, even though s_1 and s_2 share no similar terms.[2]

The second advantage of Wikimantic with coreference probabilities is its ability to do the right thing when given text that it cannot confidently interpret, whether due to its own limitations or a lack of context. Suppose we tasked Wikimantic with comparing $s_1 =$"apple" and $s_2 =$"orange". With no context to speak of, Wikimantic is forced to model s_1 using a pretty evenly weighted topic Concept C_1 that is part *Apple_Inc*, part *Apple_*$(fruit)$, and part *Apple_Records*. Likewise, it is also forced to model s_2 with an equally non-committal topic Concept C_2 which is roughly equal parts *Orange_*$(telecommunications)$, *Orange_*$(colour)$, and *Orange_*$(fruit)$. Even though both sentences are extremely ambiguous, there are many interpretations in which s_1 and s_2 are relevant. *Apple_Inc* does business with the company *Orange_*$(telecommunications)$ and *Apple_*$(fruit)$ is related to *Orange_*$(fruit)$ for obvious reasons. Since computing $P(C_1 \mid C_2)$ computes the conditional probability of every Concept in C_1 given every Concept in C_2, we are computing the coreference probability for all possible interpretations, weighted appropriately. Thus, in the absence of context terms, coreference probability comparisons will yield appropriate probability values, which is a major advantage over disambiguators that must commit to the "least wrong" interpretation of each string.

The third and final advantage is its simplicity and versatility in practical applications. From a programmer's perspective, the coreference probability calculation

[2] One may argue that term coocurrence methods could be used to detect relevance between strings that don't share similar terms. However, ambiguous terms like "steve", "jobs", and "apple" occur often in other contexts but mean different things. Since jobs is such a common term outside the context of Apple Inc, it could easily be dismissed as a "useless" term with a low TF-IDF and no strong relation to apple in particular. Nevertheless, it would be interesting to see how Wikimantic compares to an n-gram model in a formal evaluation.

function takes two strings and outputs a real number between zero and one. Little to no knowledge of Wikipedia or the generative model is actually required in order to reap their benefits, and the output format is a primitive type in almost every language. With Wikimantic, the complexities of polysemy and synonymy are completely encapsulated and hidden from the programmer without discarding useful information. This allows the programmer to operate on a higher level of abstraction, reducing the scope of the project and freeing up development time to improve performance or add new features.

BIBLIOGRAPHY

[1] Ferragina, P., Scaiella, U.: TAGME: On-the-fly annotation of short text fragments (by Wikipedia entities). In: Proceedings of the 19th ACM international conference on Information and knowledge management, pp. 1625-1628. ACM, New York (2010).

[2] Mihalcea, R., Csomai. A.: Wikify!: Linking documents to encyclopedic knowledge. In: Proceedings of the sixteenth ACM conference on Conference on information and knowledge management, pp. 233-242. ACM New York (2007).

[3] Milne, D., Witten, I.H.: Learning to Link with Wikipedia. In: Proceedings of the 17th ACM conference on Information and knowledge management, pp. 509-518. ACM, New York (2008).

[4] Ratinov, L., Roth, D., Downey, D., Anderson, M.: Local and Global Algorithms for Disambiguation to Wikipedia. In: Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, pp. 1375-1384. Association for Computational Linguistics (2011).

[5] Elzer, S., Carberry, S., Zukerman, I.: The Automated Understanding of Simple Bar Charts. Artificial Intelligence. 175(2), 526-555 (2011).

[6] Wu, P., Carberry, S., Elzer, S., Chester, D.: Recognizing the Intended Message of Line Graphs. In: Proceedings of the International Conference on the Theory and Application of Diagrams, pp. 220-234. Springer-Verlag, Heidelberg (2010).

[7] Bunescu, R., Pasca, M.: Using Encyclopedic Knowledge for Named Entity Disambiguation. In: Proceedings of the 11th Conference of the Eurpean Chapter of the Association for Computational Linguistics, pp. 9-16. EACL , Trento, Italy (2006).

[8] Fader, A., Soderland, S., Etzioni, O.:Scaling Wikipedia-based Named Entity Disambiguation to Arbitrary Web Text. In: WikiAI09 Workshop at IJCAI 2009.

[9] Li, C., Sun, A., Datta A.: A Generalized Method for Word Sense Disambiguation based on Wikipedia. In: Proceedings of the European Conference on Information Retrieval (2011).

[10] Elzer, S., Carberry, S., Chester, D., Demir, S., Green, N., Zukerman, I., Trnka, K.: Exploring and exploiting the limited utility of captions in recognizing intention in information graphics. In Proceedings of the 43rd Annual Meeting of the association for Computational Linguistics, pp. 233-230. Associate for Computational Linguistics (2005).

[11] Page, L., Brin, S., Motwani, R., Winograd, T.: The PageRank Citation Ranking: Bringing Order to the Web. Technical report, Stanford InfoLab (1999).