Wikimantic: Disambiguation for Short Queries

Christopher Boston[1], Sandra Carberry[1], and Hui Fang[2]

[1] Department of Computer Science, University of Delaware,
Newark, DE, 19716

[2] Department of Electrical and Computer Engineering, University of Delaware,
Newark, DE, 19716

Abstract. This paper 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.

Keywords: disambiguation, short queries, context, wikipedia

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 [3–5, 7]. For an encyclopedia, English Wikipedia is monolithic. It contains over 3.5 million articles which are connected 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.[3]

This paper 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 that *Visa ranks first among credit cards in circulation.* We have developed a system for identifying this high-level message [8, 9]. 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

[3] Here, we refer primarily to clear technical problems such as duplicate articles or dead links. We actually consider Wikipedia's alleged susceptibility to bias and error a boon since we expect to disambiguate correspondingly fallible user queries.

exclusively on named entities [1, 2, 5, 7]. Our experiments have shown that most queries are very short and that nouns and named entities convey only a portion of the query's full meaning. Thus a more robust method of disambiguation is required.

Our work has several novel contributions to disambiguation which are important for information 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 even 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.

The rest of the paper is organized as follows. Section 2 discusses related research. Section 3 presents an overview of the proposed method. Section 4 describes the models that select possible concepts for a query, and Section 5 discusses how the models are used to disambiguate the individual terms in the query. Section 6 presents an evaluation of the Wikimantic system and Section 7 concludes with a general summary and suggestions for future work.

2 Related Work

Bunescu and Pasca are generally credited with being the first to use Wikipedia as a resource for disambiguation [1]. 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. Though Bunescu and Pasca's work was initialy limited to named entity disambiguation, Mihalcea later developed a more general system that linked all "interesting" terms [4].

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 an impressive F-measure of 87.73[4], albeit with one caveat. Regenerating link targets is significantly easier than creating them from scratch, since the correct target must necessarily exist in Wikipedia and be particularly important to the context. Wikimantic is tasked with the more difficult problem of disambiguating all salient terms in the query indiscriminately.

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 [5, 10]. 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 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 [5]. Given a large 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 [3] 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. 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 [7]. Although their system was limited to named entities, our performance seems to be best when combining our own local and global approaches as well.

3 Method Overview

Given a sequence of terms, we seek to find a mapping from 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 $t_j \to C_i$ 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 Resignation, and "apple" would be mapped to the article for Apple Inc. Mapping "apple" to the article for the actual apple fruit would be unacceptable in that context.

Our general strategy is to first summarize the meaning of s, and then use that summary to choose the most probable mapping of terms to concepts. To summarize s, we construct a "Concept"[4] object that represents the general topic of s. The way we define and use Concept objects is based on our generative model as described in Section 4. Specifically, we begin by naively building a set of all Wikipedia articles that could possibly be referenced by terms in s. We weight each article with the product of its prior probability of being relevant and the degree to which terms in s match terms in the article. The weighted set is packaged up in a data structure we call a MixtureConcept. Once we have the weighted set, we begin searching for the best mapping from terms in s

[4] In this paper, we follow the convention that object types from our model are written in upper camel case. When we write "Concept", we refer specifically to the class of object from our model. When we write "concept", we are simply using the term as one would in every day speech.

to Wikipedia articles. We score each mapping according to the weights of the articles in the set as described in Section 5.

4 Concept Selection

4.1 Generative Model

An author encodes ideas into words and puts 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 Concepts 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 Concept 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 Concept *Apple_Corporation*'s propensity to generate such terms.

To be more precise, our generative model states that texts are 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. By incorporating knowledge of all possible Concepts and their probability of generating each term, it is possible to take a TermSequence and work backwards to find the Concepts that generated it. In order to get this knowledge, we extract a set of fundamental AtomicConcepts from Wikipedia.

AtomicConcept: An AtomicConcept is a type of Concept. Like all Concepts, an AtomicConcept has an a priori probability of being a topic Concept (denoted $P(A)$ by convention when the Concept is atomic), and probabilities $P(t|A)$ of generating term t. 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.

In order to estimate the a priori probability P(A), we look at the relative number of inter-article links that point to A's article.

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

Since Wikipedia articles link to the other articles they discuss, the fraction of incoming links is a good estimate of the likelihood that an article's subject (its AtomicConcept) will be talked about.

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.[5]

MixtureConcept: Although Wikipedia covers a wide range of topics, it would be overly simplistic to assume that each and every real world text can be accurately summarized by just one AtomicConcept. A query about Apple Inc.'s profits on the iPod Touch might better be summarized with a mixture of the AtomicConcepts $Apple\_Inc.$, $iPod\_Touch$, and $Profit\_(accounting)$. Thus we instead model the topic using a MixtureConcept which is a set of weighted AtomicConcepts. When a MixtureConcept generates a term, it randomly selects one of its AtomicConcepts to generate in its stead. The weight of an AtomicConcept tells us the probability that it will be the one selected to generate, and all weights necessarily sum to 1. Like all Concepts, a MixtureConcept M has an a priori probability $P(M)$ of being the topic, and probabilities $P(t|M)$ of generating a given term t.

$$\text{Let MixtureConcept } M = \{(w_i, A_i)|\ i = 1...n\}$$

$$\text{where } w_i = \text{the weight of } A_i \text{ in } M$$

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

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

If a term sequence s discusses Apple Inc., summarizes its profits and briefly mentions the iPod Touch, the MixtureConcept for s may look something like:

$$M_s = \{(0.5, Apple\_Inc.), (0.3, Profit\_(accounting)), (0.2, iPod\_Touch)\}$$

$$P(M_s) = 0.5{*}P(Apple\_Inc.){+}0.3{*}P(Profit\_(accounting)){+}0.2{*}P(iPod\_Touch)$$

[5] In Wikimantic, we use Microsoft n-Grams to give us P(t). Because probabilities from Wikipedia and Microsoft n-Grams each sum to 1, the sum of $P(t \mid A)$ over all t equals than 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].

To find the likelihood of the term “iPhone” in any term sequence with M_s as it’s topic, one would calculate

$$\begin{aligned} P(\text{“}iPhone\text{”}|M_s) = {} & 0.5 * P(\text{“}iPhone\text{”}|Apple\_Inc.) \\ & +0.3 * P(\text{“}iPhone\text{”}|Profit\_(accounting)) \\ & +0.2 * P(\text{“}iPhone\text{”}|iPod\_Touch) \end{aligned}$$

The key problem is how to estimate the weight w_i of each AtomicConcept. In the following, we first present a method that uses the content of a concept’s article to estimate w_i and then a second method that uses references between concepts.

4.2 Content-based Topic Modeling:

To build a MixtureConcept M that represents the meaning of a TermSequence s constituting a query, we first populate the set with AtomicConcepts and then weight the AtomicConcepts. Our base method uses the content of a concept’s article to estimate the concept’s weight in the MixtureConcept.

To construct the elements of the set, 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 the set. Any article that is disambiguated by a page whose title matches a subsequence of terms in s is also added to the set. Finally, all articles that share a disambiguation page with an article already in the set 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 the article *Steve_Jobs*.

Jobs Resigns → Matches nothing, so no articles added.

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

Once our unweighted set M is populated, it will contain a large number of candidate AtomicConcepts of varying degrees of relevance, and we rely on weights to mitigate the impact of spurious concepts. We weight each AtomicConcept according to the probability that every term in s was generated by that AtomicConcept, ignoring stopwords.

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

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

This weighting schema ensures that an AtomicConcept 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.

Once M, the estimated topic concept, has been populated and weighted, it can be used to guide disambiguation. Since AtomicConcepts in M are weighted by their propensity to generate terms in s, our first method makes the somewhat strong assumption that the probability of an AtomicConcept generating the string s is roughly equal to its probability of being the correct mapping for a term in s.

4.3 ReferenceRank: M_R

Our second method, ReferenceRank, uses references between AtomicConcepts to estimate the weights w_i. In M_R, AtomicConcepts are weighted according to the probability that they describe the sense of a given term in s, rather than the probability that they will generate a given term in s. Consider the following example where M might be misleading if it is weighted by probability of generating.

M_1 = {(0.5, Apple_Inc.),(0.5,Whole_Foods)}
M_2 = {(0.5, Apple_Inc.),(0.2,iPhone), (0.2,Apple_Safari), (0.1,iPad)}

In the text described by M_1, the topic is 50% about Apple Inc. and 50% about the grocery store Whole Foods. In the document described by M_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 M_2 than M_1. However, Apple Inc. is weighted equally in M_1 and M_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 be referred to by concepts associated with other terms 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.

Given that AtomicConcept A_1 generated a reference to another concept, 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.

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

The probability that a MixtureConcept M will generate a reference to a Concept C (denoted R_C) is just a mixture of the probabilities of the AtomicConcepts in M generating a reference to C:

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

$$\text{where } w_i = \text{the weight of } A_i \text{ in } M \text{ (Equation 1)}$$

For a TermSequence s, we compute the special MixtureConcept M_R that contains all relevant AtomicConcepts weighted by their probability of being referenced.

$$\text{Let MixtureConcept } M_R = \{(w_i, A_i)|\ i = 1...n\}$$

$$\text{where } w_i = P(R_{A_i}|M)$$

This reweighting in M_R is very similar to one iteration of the PageRank algorithm, where nodes in a graph vote for other nodes to which they link. In our case, AtomicConcepts in M vote for other AtomicConcepts in M, and the power of each vote is proportional to the weighting of that AtomicConcept in M.

5 Disambiguation

The weights of M and M_R tell us a lot about the probability that a term in s will reference a given AtomicConcept A. Since AtomicConcepts in M_R are weighted by their chances of being referenced in s, and each term we are disambiguating comes from s, one could simply use A's weight in M_R as an estimate of the probability that A is the correct sense of the term. As it turns out, M_R's reliance on Wikipedia's relatively sparse link structure causes some problems. "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 receive a vote and therefore their weights in M_R will be zero. Although M_R is useful when links are found, it must be supplemented with information from M. For this reason, the mixture $P(A|s) = (1-d)*M+d*(M_R)$ is used. The optimal value of d is determined experimentally.

In many disambiguation papers[1, 2, 4, 7], 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. For 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.

If it weren't for these conflicts, disambiguation would be simple. 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 the MixtureConcept by t in Section 4.2.

If a problematic sequence of terms like "life saver" or "new york city" is found, every possible breakdown of the sequence is disambiguated. Each breakdown yields a unique candidate set of disambiguations that is scored according to its

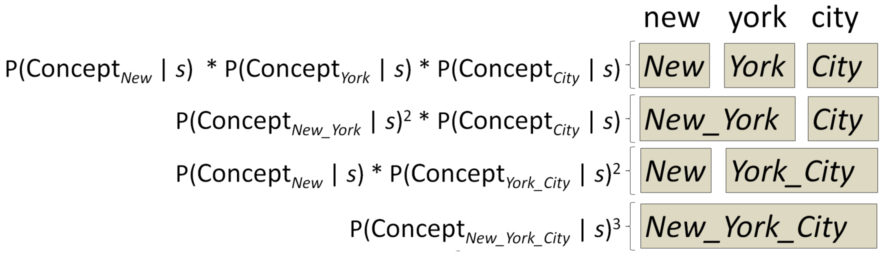
new york city
P(Concept_New | s) * P(Concept_York | s) * P(Concept_City | s)
New York City
P(Concept_New_York | s)² * P(Concept_City | s)
New_York City
P(Concept_New | s) * P(Concept_York_City | s)²
New York_City
P(Concept_New_York_City | s)³
New_York_City

Fig. 1. Product method's scoring of four possible disambiguations of "new york city".

probability of being the correct one. The scoring is calculated using either the Mixture method or the Product method.

5.1 Product Method

The Product method scores a candidate set as the product of the probabilities of each mapping of term to AtomicConcept. Figure 1 depicts the four candidate sets 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 set contains the three AtomicConcepts New, $York$, and $City$. The score of the set 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 1, where the score for the sequence "new york city" is $P(Concept_{New\_York\_City})$ to the third power.

5.2 Mixture Method

The Mixture method treats a set of possible disambiguations as a mixture of the AtomicConcepts that disambiguate terms in the set. The AtomicConcepts in M_{set} are given equal weight. Under the Mixture method, a set's score is simply equal to the average of the probability values of all Atomic Concepts in the set; 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 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).

6 Evaluation

Our system, named Wikimantic, includes four alternative methods for disambiguation: the Product and Mixture methods with M as the topic model and the Product and Mixture methods with (1-d)*M+d*M_R as the topic model.

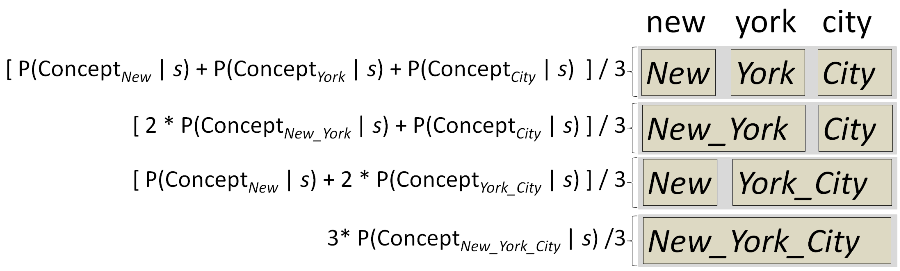
new york city
[P(Concept_New | s) + P(Concept_York | s) + P(Concept_City | s)] / 3
New York City
[2 * P(Concept_New_York | s) + P(Concept_City | s)] / 3
New_York City
[P(Concept_New | s) + 2 * P(Concept_York_City | s)] / 3
New York_City
3* P(Concept_New_York_City | s) /3
New_York_City

Fig. 2. Mixture method's scoring of four possible disambiguations of "new york city"

Each method was evaluated using 70 queries from the Trec 2007 QA track and 26 queries collected for our Information Graphic Retrieval project. The QA track was chosen because we intend to eventually incorporate Wikimantic into a larger system that operates on short grammatically correct full sentence questions, but it is worth noting that Wikimantic is in fact entirely agnostic to the grammatical structure of its input. 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 results to two evaluators and instructed them to decide for each term whether the linked page correctly described the meaning of the word as it was used in the query. The general rule was that a disambiguation was wrong if a 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.

Tables 1 and 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.

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 to be offset by a high scoring one. With the Product method, a mapping with a near-zero probability will cause the score for the entire set to be near-zero. The Product method is therefore a more conservative scoring method that favors well rounded sets over sets with some likely and some unlikely references. The exceptional performance on nouns

Performance (Nouns Only)				
	Topic Model: M		Model: (1-d)*M+d*M$_R$	
	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 1. Performance on Nouns Only

Performance (All Terms)				
	Topic Model: M		Model: (1-d)*M+d*M$_R$	
	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 2. Performance on all Non-function Words

seems to be partly due to Wikipedia's greater coverage of nouns. Additionally, Wikimantic did not incorporate a stemmer, which occasionally prevented it from recognizing matches between alternate conjugations of the same verb.

For each of the two methods described in Section 5.1, we evaluated Wikimantic using the mixture $(1-d)*M+d*M_R$ 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 correct for this.

Our results show that our system Wikimantic has very good success at disambiguating terms in short queries, even without capitalization or a priori identification of multi-word strings that should be mapped to a single concept.

7 Conclusion

In this paper, we presented a robust disambiguation method that performs well on short text fragments in which context words are scarce, that is not limited to nouns, that does not rely on correct capitalization, and that can determine when a sequence of words should be disambiguated as a single entity. Thus the approach will be useful in retrieval systems that must handle short user queries. Our disambiguation method used a two step process in which topic concepts were hypothesized via a local approach and refined with a global approach. Our experimental results show the success of the methodology and that a combination of M and M_R (ReferenceRank) has the potential to improve results, but that the disproportionate weighting of MixtureConcepts causes the top AtomicConcept to have too much voting power. Future work will explore a smoother method of weighting MixtureConcepts to overcome this problem.

Acknowledgments

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

References

1. Bunescu, R., Pasca, M.: Using Encyclopedic Knowledge for Named Entity Disambiguation. In: Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics, pp. 9-16. EACL , Trento, Italy (2006)
2. Fader, A., Soderland, S., Etzioni, O.:Scaling Wikipedia-based Named Entity Disambiguation to Arbitrary Web Text. In: WikiAI09 Workshop at IJCAI 2009.
3. 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)
4. 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)
5. 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)
6. Page, L., Brin, S., Motwani, R., Winograd, T.: The PageRank Citation Ranking: Bringing Order to the Web. Technical report, Stanford InfoLab (1999)
7. 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)
8. Elzer, S., Carberry, S., Zukerman, I.: The Automated Understanding of Simple Bar Charts. Artificial Intelligence. 175(2), 526-555 (2011)
9. 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)
10. 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)