CS273 Final Report: Entity Linking with Wikipedia

Semih Yavuz
Department of Computer Science
University of California, Santa Barbara

Isabella González
Department of Electrical & Computer Engineering
University of California, Santa Barbara

December 16, 2014

Abstract

Within a natural language text document, the intended named entities are often obscured by irrelevant entities that share the same name. Examples of named entities are first and last names, geographic locations, companies and addresses. In this project, our goal is to disambiguate named entities in a open domain text query provided by a user in natural language. In order to determine the correct counterparts of a named entities, we use Wikipedia as a reference knowledge base. The output for each named entity in a given user query is a link to a Wikipedia page of the predicted named entity. Our system reaches 78% accuracy on the test data set extracted from Wikipedia articles using hyperlink structure.

Keywords: named entity disambiguation, entity linking

1 Problem Statement

In this project, the goal is to implement an automatic named entity disambiguator from a given natural language text document. The task, also known as entity linking, is to determine the identity of entities mentioned in text. Entity linking is different from named entity recognition in the sense that it identifies the reference of the entity rather than occurrence of names. In this framework, we use Wikipedia as reference to map given named entities in a text document. Consider the following two examples:

- **Example 1:** *[Obama] is the highest point of the southwestern Shekerley Mountains, it is here that runaway slaves congregated while hiding in the forest*
- **Example 2:***One fact that might lend some credence to the theory that racism has something to do with the tenor of the attacks on [Obama] is that only one other president in our history has been the target of similar (though more subdued) personal attacks*

As can be seen from the examples above, ***Obama*** is a named entity appearing in both of the examples. However, it is corresponding to different entities in two examples above. In the first one, it refers to ***Mount Obama***, a mountain on the island of Antigua, while in the second one, it refers to ***Barack Obama***, the 44th President of the United States. Based on the examples above, the objective in this project is to determine what real world entity a given named entity in the context (text query in our case) refers to. For the unique references of real world entities, we use Wikipedia pages, which is a great information resource for this kind of purposes. Hence, for the examples provided above, we aim to come up with the following two Wikipedia pages corresponding to the query examples provided above:

- *Mount Obama:* `http://en.wikipedia.org/wiki/Mount_Obama`
- *Barack Obama:* `http://en.wikipedia.org/wiki/Barack_Obama`

In the examples above, only one named entity was considered per each query, but the problem is to determine the most likely Wikipedia reference for each named entity in the query text provided by the user.

Formally, given a vector N of named entities $N_1, \ldots, N_n$ in a text document T, the problem is to map each N_i to the correct resource from knowledge base K. In general, a named entity N_i extracted from text may correspond to several different resources in KB due to ambiguity of natural language. Let $E_{i1}, \ldots, E_{im}$ denote the candidate KB resources for named entity N_i, and E be the matrix of size $n \times m$ whose entries are E_{ij}'s. Let μ be a family of assignments from N to E such that each N_i is mapped to exactly one E_{ij}. The scoring function $S(\mu(N, E), N, T, K)$ measures the likelihood of mapping $\mu(N, E)$ based on text similarity, entity popularity, and topic consistency with respect to N, T, K. In this setting, the objective simply becomes finding an assignment μ^* with maximum likelihood by solving: $\mu = argmax_\mu$ S(μ(N,E),N,T,K)

2 Data Criterion

In this section, we provide information about the data used in this project. First, we describe our main data resource, wikipedia knowledge base, to retrieve all the information needed for entity linking task. Then, we describe extraction of training and test data sets through which the built system is optimized and tested, respectively.

2.1 Knowledge Base

Wikipedia, a notable knowledge base, will be used as information sources in this project. Wikipedia contains context information for most of its entities such as descriptive text and hyperlinks, which are useful for the purpose of disambiguation. As explained in the previous section, the goal is to map given named entities in text document to the corresponding correct reference in the knowledge base.

Wikipedia dumps are obtained from the following source:

- `http://dumps.wikimedia.org/enwiki/latest/`

In order to effectively use Wikipedia data, we use JWPL, a free Java-based application programming interface that allows to access all information in Wikipedia with the following core key features [7], [8]:

- Fast and efficient access to Wikipedia,
- Parser for the MediaWiki syntax,
- Language independent.

In this project, we use only JWPL Core which consists of Wikipedia API, and Wikipedia DataMachine:

- **DataMachine** parses and transforms the database from mediawiki format to JWPL format. The transformation produces .txt files for different tables in the JWPL database. More precisely, the steps we followed are as follows:
  - Obtained three files
    - pages-articles.xml.bz2 (11GB compressed, 50GB uncompressed)
    - pagelinks.sql.gz (5GB compressed)
    - categorylinks.sql.gz (1.5GB compressed)

    from the latest Wikipedia dump.
  - Run JWPLDataMachine by providing language, main category name, disambiguation category name parameters. This run takes 8-9 hours when 4GB memory is dedicated, and produces 11 .txt files which have the information for the different tables in the JWPL database.

  - Created a mysql database called *wikipedia*, and in which we created all necessary tables:
    - *Page*, *page_ categories*, *page_ inlinks*, *page_ outlinks*
    - *Category*, *category_pages*, *category_ inlinks*, *category_ outlinks*.
  - Finally, import all the parsed table information in .txt files produced by DataMachine before into actual database tables using **mysqlimport**.

- **Wikipedia API** is the actual Java-based application programming interface which allows us to access all kinds of Wikipedia information in the database of JWPL format that is created in the way explained above. Some of the core features it provides:
  - Retrieving Wikipedia page by title, or id.
  - Retrieving Wikipedia page's title, content, category, inlink pages, outlink pages, redirect pages, etc.
  - Querying whether a page is Wikipedia disambiguation page.
  - Querying whether a page is Wikipedia redirect page.

2.2 Training and Testing Data

In order to better optimize and test built entity linking system, we produced a data set consisting of natural language text queries and corresponding ground truth Wikipedia links for the named entities appearing in the query. One easy way of obtaining such a data set would be just going to the Wikipedia page of disambiguous named entities, and retrieve a text snippet mentioning the named entity itself, and make this text snippet and named entities it mentions a data point with ground truth. However, a data set generated this way would be just too biased to test or train the system on.

Instead, to generate a more realistic data set, we utilize the **hyperlink structure** in Wikipedia data in the following way: Choose several entities from a list of Wikipedia entities which have disambiguation pages, then navigate to another page which contains a hyperlink to the entity of interest. The text around the hyperlink to the entity of interest in the navigated page is the query that will be entered into the system. The corresponding Wikipedia page of the entity of interest is recorded as the ground truth link to it.

The natural question here would be the following: How do we obtain another page to navigate which contains a hyperlink to the entity of interest? This is where the power of link structure in Wikipedia data comes into play. If we are interested in entity e, which has a disambiguation page, then retrieving *inlink pages* pointing to e gives all the Wikipedia articles mentioning e. Incorporating this useful information, we achieve what we need.

For example, to obtain a data point for **Michael Jordan**, the professor at UC Berkeley, we navigate to Wikipedia page of *Latent Dirichlet Allocation*, and extract the following surrounding sentence mentioning the entity of interest:

LDA is an example of a topic model and was first presented as a graphical model for topic discovery by David Blei, Andrew Ng, and ***[Michael Jordan]*** *in 2003.*

Table 1: Example Queries and Ground Truth Links

Query	Ground Truth
LDA is an example of a topic model and was first presented as a graphical model for topic discovery by David Blei, Andrew Ng, and [Michael Jordan] in 2003.	`http://en.wikipedia.org/wiki/Michael_I._Jordan`
"Why Believe In You" was the first single to be taken from Scottish band [Texas]' second album Mothers Heaven.	`http://en.wikipedia.org/wiki/Texas_(band)`

3 Method

We consider the named entity disambiguation as a ranking problem. In this regard, for a given text query q mentioning named entity n, we compute a likelihood score for each candidate Wikipedia entity e corresponding to n.

So, the problem becomes a ranking problem once the candidate Wikipedia entities are determined. In this section, first we briefly explain how the candidate entities for a named entity mentioned in a query are determined which is explained in more detail in *Implementation* section. Then, we describe the method used to rank the determined candidates with respect to their likelihood to mentioned named entity.

3.1 Candidate Generation

The way we determine the candidates among which we need to disambiguate the one correct entity corresponding to entity mentioned in the query is at high-level as follows:

1. Search for disambiguation page of given named entity mention n in query q by its title.
2. If this disambiguation page exists, say p, return all the outlink pages that p points to as candidates. For example: `http://en.wikipedia.org/wiki/Michael_Jordan_(disambiguation)`
3. If not, search the page with title equal to n, with the hope that the page itself is the disambiguation page for named entity n.
4. If the result is redirected disambiguation page, say rp, return all the outlink pages that rp points to as candidates. For example: `http://en.wikipedia.org/wiki/Autumn_Leaves`.
5. If not, return the page found itself with the title equals to n, this usually happens if the named entity mention n has no disambiguity in which case we simply have one candidate which is corresponding to the page itself. For example: `http://en.wikipedia.org/wiki/David_Beckham`.

6. If no page is found when we do a title search with n, we simply cannot find any candidate for named entity n mentioned in this query q. For example: `http://en.wikipedia.org/wiki/Semih_Yavuz`

In this approach, we are not being extra careful to make sure we are not including irrelevant candidates, rather we leave this to scoring/ranking procedure, and expect it to assign poor relevance scores to such weak or unlikely candidates.

3.2 Relevance Scores

To assign the best candidate as the correct named entity, our scoring mechanism focuses on two features, entity popularity and context similarity. Given a NL text query q, a named entity mention n, and candidate Wikipedia entity set $C = \{e_1, e_2, \ldots, e_m\}$ for n. Using this notation, popularity and context similarity scores are described below.

Popularity Score

Prior popularity assumes the most prominent entity for a reference is the most probable named entity. We measure the popularity of an entity, e_i, by the number of hyperlinks mentioning that page, m. So,

$$\text{Popularity}(e_i, n, q) = \mathit{count}(m \to e_i) \tag{1}$$

However, we experimentally observed that logarithmic count of incoming links gives better results. Also, we normalize the popularity of each candidate by the sum of popularity scores of all candidates in order to get a score between 0 and 1. Hence, final popularity score we use in our application is the following:

$$\text{PopularityScore(e\_i, n, q)} = \frac{\log(1 + \text{Popularity}(e_i, n, q))}{\sum\limits_{e \in C} \log(1 + \text{Popularity}(e, n, q))} \tag{2}$$

Context Similarity Score

Context similarity measures how similar the text around the named entity in the text data and the text in the Wikipedia article are. We create two feature vectors $\vec{q}$ and $\vec{p}$ to characterize the query text q and the Wikipedia article p corresponding to candidate entity e for n. Then we use cosine similarity function between context of the query and the text of the article.

$$\text{ContextSimilarity}(e, n, q) = \frac{\vec{q} \cdot \vec{p}}{||\vec{q}||||\vec{p}||} \tag{3}$$

Now, feature vectors $\vec{q}$ and $\vec{p}$ are obtained based on *tf-idf* measure of the words appearing in the text. The purpose of using **tf-idf** measure instead of a more simple method called bag of words (BOW), which simply gives equal importance to every word occurring in the text, and represent the text just based on the number of occurrences of words in the vocabulary, is that each word is indeed not of same importance to a text document. To remedy this issue, *tf-idf* measure has experimentally been proven to produce promising results.

More precisely, let V denote vocabulary of words obtained by going through all the articles in Wikipedia and recording each distinct word w. During this process of scanning through the Wikipedia articles, we also compute document frequency of each words in vocabulary on the fly by keeping track of distinct Wikipedia documents in which a word w occurs at no extra cost. Along the way, too short, too frequent, too rare words and stop words were ignored. After this pre-processing, for a generic text document d, we represent it as a feature vector $\vec{d}$ of length $|V|$ where each word in V represented in one position of the vector. Let N be the total number of Wikipedia articles, then the importance/weight of word w in document d, which is corresponding to the value at the position w of feature vector $\vec{d}$, is the following:

$$d_w = f(w) \ln \frac{N}{df(w)}$$

where $f(w)$ is the number of occurrences of word w in text document d, and $df(w)$ is the document frequency of word w throughout the whole Wikipedia corpus. d_w is called $tf \times idf$ value of w in document d because $f(w)$ is already corresponding to **term frequency** (tf) of w, and

$$\ln \frac{N}{df(w)} = idf(w)$$

is called the **inverse document frequency** (idf) of w. This completes the explanation of how feature vectors $\vec{q}$ and $\vec{p}$ corresponding to query q and Wikipedia article p, respectively, are determined using $tf \times idf$ measure.

We apply normalization to context similarity score as well. Hence, considering 3, the final context similarity score between context of the query q containing named entity n and the text of the Wikipedia page p_i corresponding to candidate e_i is computed as follows:

$$\text{ContextSimilarityScore}(e_i, n, q) = \frac{\text{ContextSimilarity}(e_i, n, q)}{\sum_{e \in C} \text{ContextSimilarity}(e, n, q)} \tag{4}$$

Prior popularity and context similarity each provide a metric for the most likely underlying entity for a given named entity that requires disambiguation. These features can be used separately, but we generate a new score from a linear combination with a weight applied to each feature. So, considering 2 and 4, the final combined score for candidate entity e_i is:

$$\begin{aligned} Score(e_i, n, q) = {} & \alpha \times \text{PopularityScore}(e\_i, n, q) \\ & + (1 - \alpha) \times \text{ContextSimilarityScore}(e_i, n, q) \end{aligned} \tag{5}$$

The entity linking system then returns the following as a predicted Wikipedia entity for the named entity n mentioned in query q:

$$opt(e) = \operatorname*{argmax}_{e_i \in C} Score(e_i, n, q) \tag{6}$$

4 Implementation

Entire project is implemented in Java. All our external dependencies including JWPL API can be found in our pom.xml included in our source code.

4.1 Named Entity Recognition

A mentioned before, the goal of this project is to disambiguate between candidate Wikipedia links for a given named entity in a query. Hence, recognizing named entities, a.k.a NER, is not of our concern for this project. Instead, for the input text, we require user to input the text with the named entities in between brackets just like in query examples in Table 1.

4.2 Candidate Entity Generation

Given named entity n mentioned in text query q, we need to generate candidate Wikipedia entities one of which must be corresponding correct Wikipedia link for n. Candidate generation is implemented as exactly described previously in **Method** section. In this section, first, we would like to provide a couple of motivating examples which can provide the underlying intuition behind our candidate generation algorithm. Secondly, we provide some more details about the steps described in previous section.

Query 1: LDA is an example of a topic model and was first presented as a graphical model for topic discovery by David Blei, Andrew Ng, and [Michael Jordan] in 2003

Motivation 1: For this kind of example, it is easy to get all the candidate Wikipedia pages for the named entity **Michael Jordan** because there exists a disambiguation page at `http://en.wikipedia.org/wiki/Michael_Jordan_(disambiguation)` whose outlink pages immediately give us all the candidates.

Query 2: Some of Prévert's poems, such as "Les Feuilles mortes" [Autumn Leaves], "La grasse matinée" (Sleeping in), "Les bruits de la nuit" (The sounds of the night), and "Chasse à l'enfant" (The hunt for the child) were set to music by Joseph Kosma.

Motivation 2: In this example, we first search for disambiguity page of **Autumn Leaves**, but there is no such Wikipedia page. However, the Wikipedia page at `http://en.wikipedia.org/wiki/Autumn_Leaves` is itself a disambiguation page. If this is the case, we again select all the outlink pages as candidates.

Query 3: Off the court, [Kevin Durant] is a well-liked player who has donated money to various causes and is popular in Oklahoma City for his kindness toward the community.

Motivation 3: In this example, **Kevin Durant** has neither a disambiguation page nor the page itself `http://en.wikipedia.org/wiki/Kevin_Durant` is indeed a disambiguation page. Hence, in this case, we almost surely decide that named entity **Kevin Durant** has no disambiguity. Hence, it is almost surely safe to determine the page itself as the only candidate.

Considering the motivating examples above, an algorithm we came up with to select candidate entities utilizing the disambiguation pages of Wikipedia is very self-explanatory in terms of providing the underlying reasoning, and it is very effective based on experiments, although being rather simple.

Implementation Details for Candidate Generation

As can be confirmed from the source code we provided, candidate generation algorithm is very simple and consistent method implemented just as described in **Method** section. In the actual implementation in Java code, we use the following methods from JWPL Wikipedia API:

- **getPage()**: is a member of Wikipedia class, and used to retrieve the Wikipedia page with the desired **title** or **id**.
- **getOutlinks()**: is a member of Page class, and used to retrieve the pages pointed by **this** page.
- **getPlainText()**: is a member of Page class, and used to fetch the Wikipedia content of **this** page.

Having provided the functionalities above, the rest of the implementation for candidate generation is very self-explanatory based on the six items described in **Method** section.

4.3 Computation of Scores

In this section, we discuss the implementation of computation of relevance scores.

4.3.1 Popularity Score

We compute the popularity of a candidate entity e with Wikipedia page p just by using **getNumberOfInLinks()** method of **Page** class. The popularity score itself is then computed by normalizing each candidate's log popularity with the sum of log popularities of all candidates.

4.3.2 Context Similarity Score

Document Frequencies: This is an offline process we perform to compute document frequencies for words appearing in the Wikipedia corpus. In order to scan through Wikipedia articles, we use

- **getArticles()** method from **Wikipedia** class.

Once we obtain the **Iterable<Page>** object by which we are able to process Wikipedia articles for document frequency computation, a major challenge of parsing plain text content of Wikipedia pages immediately emerges because we **only want to fetch** the word stems from noisy plain text.

This parsing operation is also important during the online process of computing the similarity score between query context and the text of Wikipedia page. Since this cleaning process is a must to do, we implemented a dedicated Java class called **PageParser** in which a modified external **python** program [9] is run as a part of the cleaning process.Functionality of the resulting **PageParser** object is to fetch only the part of the text not filtered out (lets only words, and numbers in some special format to pass filtering) along with occurrence count from the text content of Wikipedia page.

Using **PageParser** and **Iterable<Page>** object returned by **getArticles()** method, as we go through the articles, we compute the document frequency of words either by updating the number of documents a word appear in or by adding the newly encountered word into the vocabulary. We store document frequencies of the words appearing in Wikipedia in the database in a table called **DocumentFrequency**.

Having stored the document frequencies in the database, given a query text q, and a candidate entities $e_1, \ldots, e_m$ with corresponding Wikipedia pages $p_1, \ldots, p_m$, compute context similarity as follows:

1. Feed text content of query q, and pages $p_1, \ldots, p_m$ in **PageParser**, and obtain distinct words along with term frequencies as output for each

2. Retrieving document frequencies from **DocumentFrequency** table in database (we can also convert document frequency values to **idf** values by single pass through the table), obtain the **idf** values for words appearing in one of q or $p_1, \ldots, p_m$.

3. Obtain feature vector representations $\vec{q}$, $\vec{p_1}, \ldots, \vec{p_m}$ of q, $p_1, \ldots, p_m$, respectively.

4. Compute context similarity between q and each page p_i using (3).

5. Compute context similarity score for each page p_i w.r.t query q using (4).

4.3.3 Disambiguation

Having computed popularity score and context similarity score for each candidate page p_i with respect to query q, compute the final score based on formula (5). Determine and return the optimal Wikipedia entity based on (6), which completes the entire pipeline for a named entity in query q. Notice that our system successfully handles the cases where there are more than one named entities in a query, just by repeating the same process for the other named entities in the query.

4.3.4 Other Details

Please refer to our source code for any other detail we have not time/space to mention in this report.

5 Results

In this section, results of the entity linking system we built are presented. **Evaluation:** For a given query q containing named entities $N_1, \ldots, N_n$, our system is considered successful on query q if it can determine the correct Wikipedia entity/link corresponding to each named entity N_i.

5.1 Tuning/Selection of combination parameter

Note that our final score function consists of linear convex combination of two scores, popularity and context similarity. Hence, we need to select a good linear combination weight so the resulting score function is effective in measuring the relevance.

Remember that we extracted a data set of NL text queries and corresponding ground truth Wikipedia links for the named entity mentions in the query. Our motivations for extracting this data set are:

- Optimize the score function
- Test the entity linking performance of whole system.

To this end, we split the extracted data into two sets, namely training/tuning examples and test examples, of equal size. Note that the combination weight α is in [0,1] because both popularity and context similarity scores are guaranteed to be in [0,1] by considering (4) and (2). Knowing that α is in [0,1], we tune α using training/tuning examples by applying two level tuning:

- **Coarse Tuning:** line searches in among [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1].
- **Fine Tuning:** line searches the neighborhood of the most promising α found by Coarse Tuning.

The plots for coarse tuning and fine tuning of α are provided in Figure 1 and Figure 2, respectively.

In coarse tuning plot, we can obviously see that prediction accuracy is the best for the values of α in [0.6, 0.7]. Hence, we try to zoom in that interval and fine tune the value of α in the neighborhood of 0.7. We search α values in [0.55, 0.75] interval with 0.01 apart, but see that more resolution does not help much in this case because the best accuracy for training set is still obtained for the α values in [0.6, 0.7] interval. Also, as can be seen Figure 2, accuracy is the same for all the values of α in [0.6, 0.7] interval. Hence, we finalize the following three different values for α to use while testing the system: 0.6, 0.65, 0.7. According to training data, these values are the representatives of the best interval for α in terms of accuracy results. In addition to these, we also add 0.8 just to try to see if selecting α from the best interval overfits the training data. In addition to these, we also have two baseline performance measures that we compare our results with:

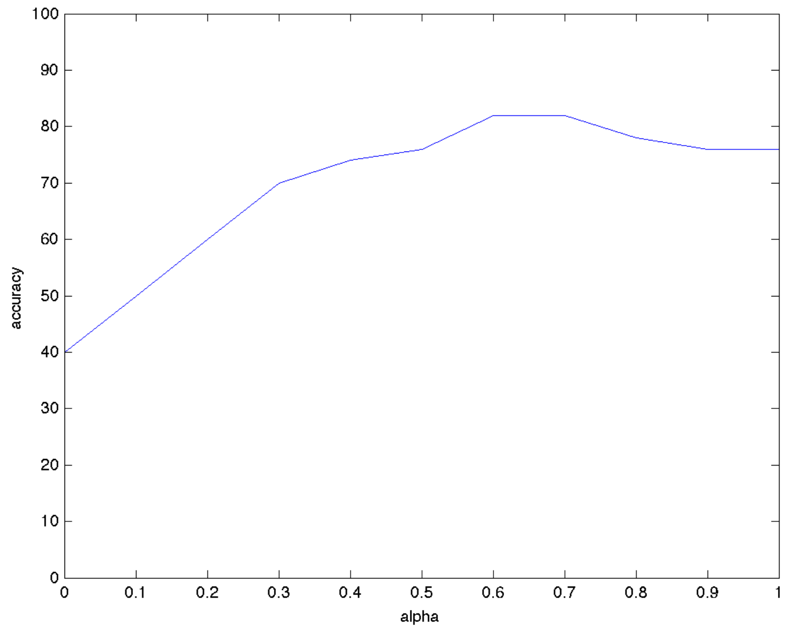

Figure 1: Coarse Tuning

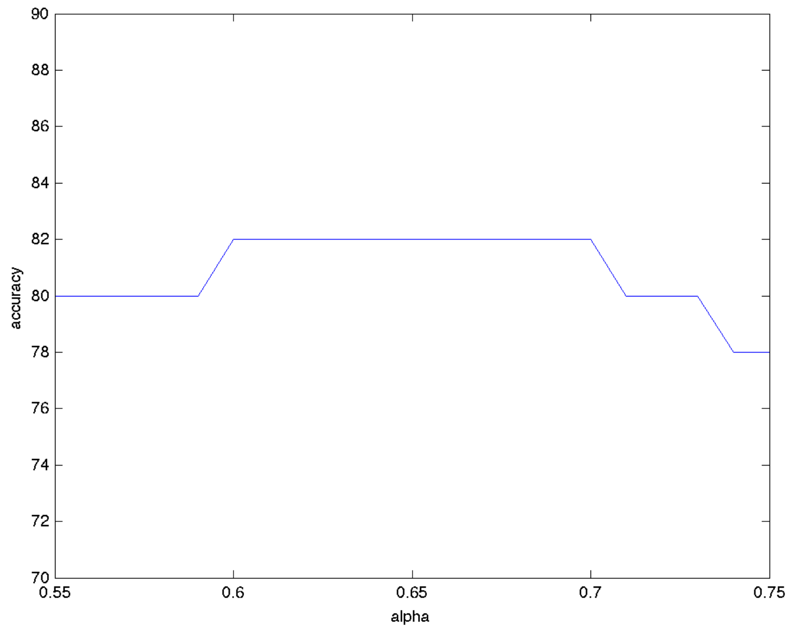

Figure 2: Fine Tuning

Baseline A for selecting α: If we did not use any examples for tuning/training the value of combination parameter α, it would be set as: $\alpha = 0.5$

Baseline B for scoring function: One most simple baseline for scoring function would be to use popularity score as the only component of score function, which is exactly corresponding to $\alpha = 1$ case in our system.

5.2 Test Results

Below, we present a table in which we test performance of our system for best α candidates based on training/tuning data set, and compare it with the additional three α values we selected for baseline performance measures and also for attempting to check if we can observe an overfitting.

Dataset	α =0.6	α =0.65	α =0.7	α =0.8 (O)	α =0.5 (A)	α =1 (B)
Test Data	72%	72%	74%	74%	70%	64%
Entire Data	76%	76%	78%	76%	73%	70%

As can be seen from the table above, tuning of α based on tuning/training data allows our system to perform better than both baseline A with $\alpha = 0.5$ and baseline B with $\alpha = 1$. On the other hand, two of the values 0.6, and 0.65 are beaten by another randomly selected α value 0.8, which shows us that selection of α based on tuning examples may overfit to its data points, especially when there is a few examples for training. This indeed leads to another point/discussion: since the number of training/tuning examples we have is very few, we cannot observe a leap in the relative performance w.r.t other random or baseline performances.

Accuracy results for the same α values selected to test the system with are also provided just as some more additional results in the second row of the table above. As can be seen, scoring function based first three α values 0.6, 0.65, and 0.7 tuned based on training examples outperform both baseline A and baseline B, and also perform at least as good as the score function with a random $\alpha = 0.8$.

5.3 Examples of Successful and Unsuccessful Entity Disambiguation Results

- For all the results presented in this section, we use the score function with convex linear combination parameter $\alpha = 0.7$:
- In the next page, we present actual results that our system generated for some selected queries in test data.
- In Table 2 in the next page, we provide some examples from test data for which our system successfully disambiguate the correct Wikipedia entity for a given named entity contained in a text query.

Query	Actual Entity	Predicted Entity
Necker's dismissal on 11 July 1789, provoked by his decision not to attend Louis XVI's speech to the Estates-General, eventually provoked the storming of the [Bastille] on 14 July.	`http://en.wikipedia.org/wiki/Storming_of_the_Bastille`	`http://en.wikipedia.org/wiki/Storming_of_the_Bastille`
From 2001 to 2004 Burt Kwouk provided voice-overs on the spoof Japanese betting show [Banzai] and subsequently appeared in adverts for the betting company, Bet365.	`http://en.wikipedia.org/wiki/Banzai_(TV_series)`	`http://en.wikipedia.org/wiki/Banzai_(TV_series)`
[Audible] is a seller and producer of spoken audio entertainment, information, and educational programming on the Internet.	`http://en.wikipedia.org/wiki/Audible.com`	`http://en.wikipedia.org/wiki/Audible.com`
Mel Ziegler and his wife Patricia Ziegler were the founders of [Banana Republic].	`http://en.wikipedia.org/wiki/Banana_Republic`	`http://en.wikipedia.org/wiki/Banana_Republic`
Goalkeeper [Michael Jordan], who started his career as a trainee at Arsenal, has also now agreed a one-year contract at Saltergate	`http://en.wikipedia.org/wiki/Michael_Jordan_(footballer)`	`http://en.wikipedia.org/wiki/Michael_Jordan_(footballer)`

Table 2: Successful Entity Disambiguations

- Lastly, in Table 3 below, we present some examples from test data for which our system fails to disambiguate the correct Wikipedia entity.

Query	Actual Entity	Predicted Entity
The five-year voyage of Charles Darwin on the ship HMS [Beagle] established him as an eminent geologist.	`http://en.wikipedia.org/wiki/HMS_Beagle`	`http://en.wikipedia.org/wiki/List_of_ships_named_HMS_Beagle)`
In October 2008, Martin Weyl turned an old garbage dump near [Ben Gurion] International Airport, called Hiriya, into an attraction by building an arc of plastic bottles.	`http://en.wikipedia.org/wiki/Ben_Gurion_Airport`	`http://en.wikipedia.org/wiki/Ben-Gurion_House`
In 1935 Hawks made [Barbary Coast] with Edward G. Robinson and Miriam Hopkins.	`http://en.wikipedia.org/wiki/Barbary_Coast_(film)`	`http://en.wikipedia.org/wiki/Barbary_Coast,_San_Francisco`

Table 3: Failure Entity Disambiguations

References

[1] D. Milne and I. H. Witten. Learning to Link with Wikipedia. In CIKM, 2008.

[2] T. Zesch, C. Müller and I. Gurevych. Extracting Lexical Semantic Knowledge from Wikipedia and Wiktionary. In LREC 2008.

[3] D.B. Nguyen, J. Hoffart, M. Theobald, and G. Weikum. AIDA-Light: High-Throughput Named-Entity Disambiguation. In LDOW 2014.

[4] J.Hoffart et al. Robust Disambiguation of Named Entities in Text. In EMNLP 2011.

[5] A. Alhelbawy and R. Gaizauskas. Graph Ranking for Collective Named Entity Disambiguation. In ACL 2013.

[6] R. Usbeck et al. AGDISTIS - Graph-Based Disambiguation of Named Entities using Linked Data. In ISWC 2014.

[7] https://code.google.com/p/jwpl/

[8] T. Zesch, C. Muller, I. Gurevych. Extracting Lexical Semantic Knowledge from Wikipedia and Wiktionary. LREC 2008.

[9] G. Attardi and A. Fuschetto. Part of Tanl. 2009.