

# LADS: Rapid Development of a Learning-To-Rank Based Related Entity Finding System using Open Advancement

Bo Lin, Kevin Dela Rosa, Rushin Shah, Nitin Agarwal

Language Technologies Institute

School of Computer Science

Carnegie Mellon University

5000 Forbes Ave., Pittsburgh, PA 15213 USA

{bolin,kdelaros,rnshah,nitina}@cs.cmu.edu

## ABSTRACT

In this paper, we present our system called LADS, tailored to work on the TREC Entity Track Task of Related Entity Finding. The LADS system consists of four key components: document retrieval, entity extraction, feature extraction and entity ranking. We adopt the open advancement framework for the rapid development and use a learning-to-rank approach to rank candidate entities. We also experiment with various commercial and academic NLP tools. In our final experiments with the TREC 2010 dataset, our system achieves the fourth rank compared to the fifteen teams who participated in TREC 2010.

## Categories and Subject Descriptors

H.3.3 [Information Storage and Retrieval]: Information Search and Retrieval – *information filtering retrieval models, search process, selection process.*

## General Terms

Algorithms, Design, Experimentation

## Keywords

Named Entity Recognition, Learning to Rank, Information Retrieval

## 1. INTRODUCTION

We focus on the task of related entity finding, as defined by the TREC 2011 Entity Track. This is essentially an entity-oriented web search task. As input, we are given “topics”, which consist of the name of an input entity, the type of the target entity, and a narrative describing the nature of the relationship between entities. There are four different target entity types: organization, location, person, and product. The goal of the task is to find homepages, represented by their ClueWeb09 id [2], for target entities for a given topic. This is a challenging task due to several factors; one being that web search engines are optimized to find documents and not entities, and another being that it is harder to precisely convey the semantics of a relation to a search engine. Our approach to this task is notable for two reasons: Firstly, in

order to achieve fast iteration and the best possible results, we used an open advancement approach and designed a modular architecture that allows us to easily plug in different NLP components, including off-the-shelf commercial ones. Secondly, we made use of a rich feature set to rank candidate entities, and experimented with a Learning to Rank (LETOR) approach to combine these features in a sophisticated manner. We were able to create a system that achieved results close to the best published numbers; with the advantage of requiring very little development time.

## 2. RELATED WORK

The TREC evaluations have been the pre-eminent venue for advancements in the field of information retrieval. The TREC Entity track was added a few years ago with the aim of encouraging entity oriented research. The Related Entity Finding task is currently the main task in this track. The Clueweb 09 corpus that we used was compiled with the aim of allowing researchers to develop and test our system on a complete snapshot of the web, as it existed in 2009. There has also been a lot of prior work in developing the open advancement software engineering approach, particularly in the field of question answering.

## 3. DATASET AND EVALUATION

### 3.1 Dataset

We use topics from the 2010 REF data set. This data set consists of 50 topics, with a heavy skew towards the “organization” target entity type (60%). Of these topics, 47 were ultimately judged, with homepages being pooled (depth 20) from all of the participants in the 2010 evaluation [8].

Homepages are retrieved from the Clueweb 09 corpus, and acceptable homepages are ranked as either being “primary”, or pages that are devoted and in control of the entity (i.e. <http://www.metallica.com/>), and “related”, or pages that are devoted but not in control of the entity (i.e. <http://www.last.fm/music/Metallica>). For the 2010 offering of the task, Wikipedia pages are disallowed as homepages. An example query from the 2010 data set, in XML format is in figure 1:

Figure 1: Example TREC 2010 Entity Track Topic

```
<query>
  <num>25</num>
  <entity_name>U.S. Supreme Court</entity_name>
  <entity_URL>clueweb09-en0012-87-
19363</entity_URL>
  <target_entity>organization</target_entity>
```

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

EOS, SIGIR 2011 Workshop, July 28, 2011, Beijing, China.

Copyright is held by the author/owner(s).

```
<narrative> From what schools did the Supreme Court
justices
    receive their undergraduate degrees?
</narrative>
</query>
```

## 3.2 Evaluation

We use NDCG@R as our primary metric to evaluate our results. We set the gain to 1 for relevant pages, and 3 for primary pages. In addition, we also report the number of primary and relevant pages retrieved, and a few other standard information retrieval metrics such as R-precision (Rprec), mean average precision (MAP), and precision at 10 (P@10).

## 4. ARCHITECTURE AND COMPONENTS

### 4.1 Architecture

As mentioned earlier, one of our key objectives in designing our system was to have the ability to develop parts independently and combine them quickly, and ensure quick experimentation and frequent integration of components. To this end, we designed our system to consist of interchangeable components with common interfaces, and a specific pipeline set dynamically via configuration files. Our pipeline contains the following abstract base components:

- Query Analysis: Performs an initial analysis of the input topic queries
- Document Retrieval: Searches for documents based on input from the query analysis
- Entity Extractor: Extracts candidate entities from the retrieved document pool
- Feature Extractor: Extracts features for each candidate entity
- Entity Ranker: Ranks entities based on their feature values
- Homepage Retrieval: Finds homepages for the ranked entity list

Each of these components is described in detail in the following subsection.

### 4.2 Components

#### 4.2.1 Query Analysis

We use OpenEphyras [5] to extract keywords from query narratives and supplied this information to subsequent components in the pipeline. We do not perform more sophisticated processing with query analysis because of its limited value in increasing the overall accuracy of our system.

#### 4.2.2 Document Retrieval

We experiment with using web-based search as well as Indri-based local search for document retrieval. For both cases, we vary parameters such as the number of documents retrieved, and the use of full query narratives versus only keywords.

##### 4.2.2.1 Web Search

We wrap two different commercial web search engines, including Yahoo BOSS [7] (build your own search service), and Microsoft Bing [4]. For our final system, we use Yahoo BOSS since it was more reliable and flexible with respect to behavior and API rate limitations; though anecdotally, the results from both services are quite similar.

##### 4.2.2.2 Local Search

We use a JNI-based Indri interface [6] that searches the ClueWeb09 category B corpus (a set of 50 million high-quality English language webpages, including all of Wikipedia) [2]. We do not use any of Indri's sophisticated query operators, as we are interested in treating the web and local search engines as black box search engines with minimal changes to the queries provided as input.

##### 4.2.3 Entity Extraction

We adopted and experimented with state-of-art named entity extraction systems, including academic [11] and commercial products [1], to extract candidate entities from the retrieved documents for feature extraction and entity ranking. The input for the entity extraction component is HTML web page and the desired output is the set of candidate entities extracted from the web page. We implemented and investigated various heuristics to improve such webpage-based entity extraction.

##### 4.2.3.1 Stanford NER

Stanford Named Entity Recognizer (Stanford NER) is a commonly used academic tool for named entity recognition known for the accuracy of its results. It incorporates conditional random field-based classifier to label the named entities in the text. Yet Stanford NER is originally designed and trained with plain text without HTML decorations. Thus we made several efforts to adapt the Stanford NER for our purpose of extracting entities from HTML web pages.

Firstly, we incorporated a standard HTML text extractor, JSoup [3], to perform text extraction from HTML web pages. It mainly removes the HTML tags, formatting information and meta-data. Secondly, we varied the system by incorporating structure information in the HTML web pages. The intuition is that simply removing the HTML tags in the web pages eliminates the structured information associated with them. Yet such information might be useful for considering the candidacy of the entities. By applying Stanford NER directly on the extracted text, we failed to use the structure information. Thus we decided to use the readily available structure information in the HTML web pages - the HTML elements (e.g. "body", "title", "anchor text", "table" etc.). We hypothesize that since many queries are targeted towards a list of entities thus the candidate entities are more likely to appear in the lists or tables inside the HTML web pages. Thirdly, we improved the system by using the heuristics of injecting delimiters to the HTML web page. In our early error analysis, we found that the removal of the HTML tags in the HTML text created this problem. HTML web pages sometimes use the HTML elements as text delimiters; however, our HTML text extractor removes all these delimiters to produce the plain text for Stanford NER to work with. Since Stanford NER is trained on News texts with numerous text delimiters such as punctuation marks, it fails to separate the neighboring entities in our case. We then tested our heuristics to inject artificial delimiters to the HTML web pages. While there are many artificial delimiters we could use, we chose the punctuation marks "," since it has less ambiguity than "." (which can be considered as a mark for initial other than a sentence break) or "-" (which can be used to connect two words). We inserted the punctuation marks "," before every occurrence of HTML tag-closing statement "</" in the HTML text before the HTML text extraction. Fourthly, we attempted and experimented with several heuristics to filter ill-formed entities in order to improve the system. We designed and tried three different filtering

heuristics to remove the ill-formed candidate entities. The first filtering heuristics consider the number of characters in the surface form of the token, discard the entity if it is shorter than a threshold value. The second filtering heuristics consider the number of tokens in the entity, discard the entity if the number of tokens is less than a minimum value or larger than a maximum value. We tried a few different values for the thresholds in these two filters and eventually picked 4 for the minimum number of characters in the entity surface form, 1 for the minimum number of tokens and 5 for maximum number of tokens in the entity. The third filtering heuristics consider the tokens in the entity, if they consist of only function words, the entity is discarded. The function word list is obtained from the open sourced question answering tool OpenEphyra [5].

#### 4.2.3.2 AlchemyAPI NER

AlchemyAPI Named Entity Extractor (NEE) is one of the state-of-art commercial tools for named entity extraction. We conducted an extensive survey of both state-of-art academic and commercial tools and found that AlchemyAPI NEE is one of the best among them and provides the best balance of extraction accuracy and processing speed.

AlchemyAPI NEE is capable of identifying people, companies, organizations, cities, geographic features, and other typed entities within HTML text, plain text, or web-based content. It supports entity disambiguation which links the extracted entity to its corresponding entry in external database including DBpedia, Freebase, OpenCyc etc.. It also supports entity type identification for 34 different entity types such as automobile, city, facility.

We implemented AlchemyAPI in our system with manually created mapping between its 34 entity types to the 4 entity types in TREC entity track.

#### 4.2.4 Feature Extraction

We investigated a wide set of features that we thought might be good indicators of how relevant a candidate entity is to the query.

- **Frequency Based Features**

These features are based on the number of times a candidate entity shows up in search results. In particular, we count

- F1 - The total number of occurrences of a candidate entity in all the search results
- F2 - The number of unique search results that an entity occurs in

- **Density Based Features**

These features consider the quality of web pages that a candidate entity occurs in, as judged by the number of other candidate entities in these web pages. Specifically, for a particular candidate entity  $E_C$  we count:

- D1 - The number of all entities in a webpage that  $E_C$  occurs in (summed over all the search results that  $E_C$  occurs in).
- D2 - The number of unique entities in a webpage that  $E_C$  occurs in (summed over all the search results that  $E_C$  occurs in).

- **Proximity Based Features**

These features are based on the concept of ranking entities according to how closely they occur to query keywords in the retrieved web pages. We use two such features:

- P: The minimum distance between any query keyword and a candidate entity  $E_C$  in a webpage (averaged across all the search results that  $E_C$  occurs in).
- $P_N$ : Product of cumulative proximity between keywords and entities in retrieved documents and number of such documents. Formally,

$$Score(c) = numDocuments(c) \times \sum f(dist(k,c), \delta)$$

where  $dist$  is the proximity and  $f$  is any aggregate function such as summation or maximum.

- **Semantic Similarity Based Features**

We evaluate semantic similarity between words using the Jiang Conrath metric [12], which considers the WordNet distance between two strings. We apply this metric to count the following specific features:

- S1: Similarity between the query narrative and the snippet of a search result (Averaged across all search results for an entity  $E_C$ ).
- S2: Similarity between the keywords in the query and the type of the candidate entity
- S3: Similarity between the keywords in the query and the Freebase description of the candidate entity

- **Average Rank of Web Pages**

We consider the average rank of the web pages that a candidate entity  $E_C$  occurs in, with the intuition that entities that appear in higher ranked pages on average are more likely to be relevant. We refer to this feature as AvgRank subsequently.

#### 4.2.5 Entity Ranking

We noticed that almost all the systems that have participated in the TREC entity track in previous years used some sort of linear weighting scheme to combine different features in order to rank entities, and one of our motivations was to use a learning to rank (LETOR) approach for this task and see whether such an approach would outperform traditional linear weighting schemes. We therefore developed an SVM model to rank candidate entities.

A significant problem in using an SVM for this task is finding gold standard data to train such a model, because the relevance judgments from the previous years that are available to us are in the form of binary values that indicate whether an entity is relevant or not, while our desired output from the SVM ranker is a real-valued score for each candidate entity. To get around this problem, we train an SVM using this binary-formatted data, but while testing on new instances, we have it output a score between 0 and 1, indicating how likely an entity is to be relevant.

More precisely, we train on a subset of the queries, and during

**Table 3: Effect of various improvements on Stanford NER**

	# Relevant	# Primary	P@10	NDCG@	MAP	Rprec
Stanford	31	108	0.0681	0.096	0.0551	0.078
Stanford-Injection	<b>50</b>	<b>160</b>	<b>0.1</b>	<b>0.1306</b>	<b>0.0792</b>	<b>0.1093</b>
Stanford-Minmax	30	108	0.0702	0.0949	0.0551	0.078
Stanford-Minmax-Keyword	27	108	0.0766	0.0995	0.0582	0.0781

training we check if a candidate entity for a particular query is present in the relevance judgment file and marked as a primary or relevant entity. If so, this constitutes a positive instance for the SVM, and if not, a negative instance. We encounter many more negative instances than positive ones, so we keep a fixed ratio of negative instances for each positive instance found, and throw away the rest of the negative instances. We then perform a grid search to find the optimal parameters for training and subsequently train our SVM. During testing, we have it output a relevance score for each candidate entity and classify entities according to this score. We use the LibSVM [9] library to implement SVMs.

#### 4.2.6 Homepage Retrieval

We started with a naive homepage retrieval strategist that returns the first page from a search engine which is not a Wikipedia page, since the Wikipedia domain is explicitly disallowed for the TREC Related Entity Finding task. We picked this strategy because search engines like Google & Yahoo do a very good job at returning homepages, or at least very devoted and informative pages, for entities & topics, and in fact this turned out to be a surprisingly difficult strategy to beat. One change that was a significant improvement over this naive strategy was using a list of “blacklisted domains” for use in filtering irrelevant homepages. The domains we filtered were primarily content farms, such as “answers.com”, “about.com” and “mahalo.com”. Content farms are typically web pages that have high concentrations of text that are specifically designed to be ranked highly by web search engine algorithms and are thus unlikely to be considered a relevant or primary page with respect to the TREC Entity task. In other words, instead of rejecting Wikipedia pages from consideration, we reject any page from a blacklisted domain.

## 5. EXPERIMENTS AND ANALYSIS

In this section we describe the results of each of our component experiments, and provide an analysis of the experimental results.

### 5.1 Document Retrieval

We experimented with using both full query narratives and keywords only, and found that both strategies yielded the same number of related pages. Using the full narrative had a slight edge in retrieving more primaries, but NDCG@R was higher for the key word strategy. This is due to the fact that the key word search returned slightly better quality documents, which were ultimately beneficial in entity ranking.

We also used both web-based (Yahoo, Bing) and local (Indri) search engines, and found that web search clearly out performs

**Table 1: Performance with full query narrative versus using keywords only**

	# Relevant	# Primary	P@10	NDCG@	MAP	Rprec
Full Narrative	40	183	0.1021	0.1176	0.072	0.0861
Key Word	40	176	0.1106	0.1182	0.0662	0.0909

**Table 2: Performance with web search (Yahoo) versus Indri for document retrieval**

	# Relevant	# Primary	P@10	NDCG@	MAP	Rprec
Indri	37	154	0.1	0.0978	0.0513	0.0855
Web Search	40	176	0.1106	0.1182	0.0662	0.0909

the Indri search on all counts, since it gets significantly better quality pages which resulted in more related and primary pages.

### 5.2 Entity Extraction

We used heuristics and filtering techniques to improve Stanford NER on web-page entity extraction. The baseline run “Stanford” is the same as in previous experiment by directly applying Stanford NER on the text extracted from the entire web-page. Other runs show the result from using Stanford NER together with the delimiter injection heuristics (“Stanford-Injection”); and using Stanford NER together with the entity length filtering heuristics which filters the entity with too many / few tokens or too few characters (“Stanford-Minmax”); and using Stanford NER with the entity length filtering heuristics and keyword filtering heuristics which filters the entity containing only function words from the OpenEphyra’s function word list (“Stanford-Minmax-Keyword”). The following table shows the measures of performance from these runs.

As observed, delimiter injection outperforms others by 30% in all the measures, which confirms our early error analysis that there are around 10%-20% ill-formed entities are related to the removal of delimiters in HTML text extraction. The other filtering techniques also helped improve P@10, NDCG@R and MAP relatively by 5%-10%. Eventually, we selected “Stanford-Injection” and “Stanford-Minmax-Keyword” as the two competitive settings for Stanford NER in our final runs.

We also compared performance of our system with the use of Stanford NER versus AlchemyAPI NEE. The performances of the two runs differ in different metrics. In terms of P@10 and Rprec, the “Best-Alchemy” outperforms the “Best-Stanford” relatively by 20%-30%. However, in NDCG@R and MAP, “Best-Stanford” performs slightly better than “Best-Alchemy”.

Also with respect to NDCG@R, the best run using Stanford NER

**Table 4: Performance with Stanford NER versus AlchemyAPI**

	# Relevant	# Primary	P@10	NDCG@	MAP	Rprec
Stanford NER	48	159	0.1064	<b>0.1352</b>	<b>0.0815</b>	0.1163
AlchemyAPI	36	<b>168</b>	<b>0.1362</b>	0.1339	0.08	<b>0.1302</b>

with delimiter injection is the best run of our entire system. We further investigated this problem and discovered that “Best-Stanford” produced more relevant results (as of “# Relevant”) and more total correct results (as the sum of “# Relevant” and “# Primary”) in terms of number of home pages. But “Best-Stanford” suffered in the primary results it returned (as of “# Primary”) thus in the measures of P@10 and Rprec. We hypothesized that by using Stanford NER, we obtained more entities than using AlchemyAPI NEE. However, the entities returned by AlchemyAPI NEE is more accurate than that by Stanford NER, which might have significantly affected the down-stream homepage retrieval component.

### 5.3 Feature Extraction

We tested the relative performance of our extracted features, by using each feature individually to rank the candidate entities.

We notice a number of interesting trends in these results. For the frequency based and density based features, we find that counting all occurrences of an entity in a document instead of unique occurrences of the entity (i.e. F1 and D1 instead of F2 and D2)

leads to slightly better performance. Also, the frequency based features perform better than the equivalent density based ones.

The proximity feature  $P_N$  is the best performing feature on almost all metrics; notably, it does better than the pure proximity feature  $P$  and all the semantic similarity features  $S1$ ,  $S2$  and  $S3$ . Surprisingly, AvgRank (the average rank of web pages), which is quite a naive feature, is the next best performing feature. Amongst the semantic similarity based features,  $S1$  (similarity between query narrative and candidate webpage snippet) performs reasonably well, but  $S2$  (similarity between the given target type and the candidate entity type) and  $S3$  (similarity between given target type and candidate entity freebase type) both give very poor performance. We believe that while type similarity information might still be valuable, the way in which we were interpreting semantic similarity might be flawed and hence cause such poor results.

## 5.4 Entity Ranking

We tried different variants of our SVM Ranker: We trained one variant  $V1$  using the default parameters supplied by the LibSVM [9] tool, and another one  $V2$  using tuned parameters that we calculated using a grid search. We also used a feature selection tool and trained another variant  $V3$  that used only the features suggested by this tool, namely  $F1$ ,  $D2$  and  $P$ .

As expected, we find that the model that uses the tuned parameters performs better than the one that uses default parameters. Surprisingly however, we find that neither of the models performs as well as just using the proximity feature  $P_N$  alone (results using individual features for ranking are detailed in the previous section). This contradicts our initial assumption that the use of SVMs for ranking entities would produce better performance than using any individual feature alone. We also find that the model  $V3$  that uses the subset of features suggested by the feature selection tool doesn't perform as well as the tuned model  $V2$  that uses all the features in the refined feature set. This suggests that feature selection is not very effective for this problem.

As a baseline, we also trained a linear combination model that simply combined all of our features (except  $S2$  and  $S3$ , which performed quite poorly on their own), with equal weights. Surprisingly, this model performs slightly better than the tuned SVM model using the same feature set. This negates our assumption that SVMs would be a more effective way to combine different types of features for ranking than linear combination.

However, none of the above models perform as well as simply using the proximity feature  $P_N$  alone to rank candidate entities (results using individual features for ranking are detailed in the previous section). We tried running an exhaustive parameter sweep, since this might have yielded a set of weights that performed better than the feature  $P_N$  alone, but the total number of possible models was exponential in the number of features, and since each model required re-running at least the homepage retrieval component of our system, we quickly encountered API rate limits (even when we cached previously found web pages). Ultimately, this parameter sweep proved intractable. It is an interesting point to note that even though linear combination rankers are conceptually much simpler than SVMs, optimizing their parameters requires more resources for our system than optimizing the parameters of our SVM model.

**Table 5: Performance of the system with individual features used to rank candidate entities**

	# Relevant	# Primary	P@10	NDCG@	MAP	Rprec
F1 (Frequency)	34	147	0.1064	0.1146	0.0665	0.1034
F2 (Frequency)	34	147	0.1064	0.114	0.0658	0.1032
D1 (Density)	32	142	0.0957	0.1071	0.0607	0.0855
D2 (Density)	32	141	0.0957	0.1063	0.0604	0.0861
P (Proximity)	33	147	0.0957	0.112	0.0628	0.0847
S1 (Semantic)	31	145	0.1106	0.1121	0.0646	0.1065
S2 (Semantic)	33	138	0.0255	0.0392	0.0212	0.036
S3 (Semantic)	32	136	0.0321	0.0401	0.0228	0.0341
AvgRank	33	143	0.1	0.1164	0.062	0.1085
C (Combination)	32	150	0.1128	0.1226	0.0696	0.1123

**Table 6: Performance with various SVM models for ranking candidate entities**

	# Relevant	# Primary	P@10	NDCG@	MAP	Rprec
V1 (un-tuned)	30	141	0.0915	0.1025	0.0583	0.0964
V2 (tuned)	33	141	0.1	0.1073	0.0651	0.0924
V3 (feature-selection)	32	141	0.0934	0.1041	0.0603	0.0945
Linear	33	145	0.1125	0.1187	0.0665	0.1111
Combination						

**Table 7: Improvement due to perfect homepage retrieval**

	NDCG@R	MAP	Rprec
Regular H/P Retrieval	0.1182	0.0662	0.0909
Perfect H/P Retrieval	0.1976	0.1386	0.141

## 5.5 Homepage Retrieval

The evaluations and experiments of LADS system shown in previous sections are inherently imperfect because the only homepages considered relevant according to the relevance judgment file from the 2010 TREC entity track are based on the pooled results submitted by the participants to the track. Results other than the ones existed in the relevance judgment file will not be considered. Our system often produces the correct entities and obviously relevant homepages but failed to match the ones included in the track's judgment file. Since we are more interested in the problem of discovering related entities, and not necessarily their homepages, we conduct an experiment where we assume we have a perfect homepage retrieval system which gives exactly the same homepages as those in the relevance judgment file. If an entity retrieved by the system match exactly some entity in the relevance judgment file, we consider it relevant regardless of its homepage. We can see from Table 7 that this results in a dramatic improvement in the performance of our system. We use this modification as part of our system when making our final runs, the results of which are reported in the following section.

## 6. FINAL RESULTS

We picked the following configurations to be our final runs, and their results in comparison with the best and median automatic results from the TREC 2010 Entity task evaluation are reported in Table 8.

- **Run 1:** Baseline: Yahoo, 40 documents, AlchemyAPI, Ranking using  $P_N$ , no blacklist filtering

- **Run 2:** Run 1 with SVM ranker instead of proximity feature  $P_N$
- **Run 3:** Run 1 with the blacklist homepage filter applied
- **Run 4:** Run 1 with Stanford NER instead of Alchemy NEE.

Our results are consistently above the median value, with the best results being for Run 1, and would place us around the 4<sup>th</sup> place among all the teams.

## 7. DISCUSSION

**Table 8: Best final runs of our system**

	NDCG@R	MAP	Rprec
2010 Best System	0.3694	0.2726	0.3075
2010 4th Best System	0.1696	0.0953	0.1453
Run 1	<b>0.2036</b>	<b>0.1406</b>	<b>0.1687</b>
Run 2	0.1754	0.1096	0.1322
Run 3	0.1885	0.1272	0.1276
Run 4	0.1926	0.1188	0.1393

Some of the main advantages of our system include the modular and pluggable software engineering approach we used to implement it, our usage of commercial IR and NLP tools, the diverse feature set we implemented to rank candidate entities, and our use of a learning-to-rank (LETOR) approach in the form of our SVM Ranker. Our architecture allowed us to iterate quickly and independently even though there are a lot of moving parts in our system, and it also allows for easy improvements to our system in the future. We didn't achieve the improvements that we expected from using SVMs to rank candidate entities, but the feature set we used to rank candidate entities is diverse enough that we are optimistic of obtaining better performance in the future as we work on better ways to combine these features, including potentially more sophisticated LETOR approaches than our current one. We also found that while commercial NLP tools may not achieve state-of-the-art performance compared to academic systems, they are often more robust and have certain other advantages. For example, AlchemyAPI, the commercial NER system that we used for entity extraction, has a more fine-grained type system than academic NER systems such as Stanford NER. Consequently, we adapted a number of such tools within our system.

## 8. CONCLUSION

In terms of future work, we plan to investigate ways to better incorporate the source entity in our system, using structured queries for Indri search, using additional sources of information for homepage retrieval besides just surface forms of entities, and looking for alternative ways to frame entity ranking as a learning-to-rank task. We also intend to open-source our system (after

submission to TREC), and we hope that it will be valuable to other research groups wishing to work in the field of Related Entity Finding, or related areas such as Question Answering.

## 9. ACKNOWLEDGEMENTS

We would like to thank Professor Jamie Callan at Carnegie Mellon University for his support and guidance.

## 10. REFERENCES

- [1] Alchemy API. [Online]. Available: <http://www.alchemyapi.com>
- [2] ClueWeb09. [Online]. Available: <http://boston.lti.cs.cmu.edu/clueweb09/wiki/tiki-index.php>
- [3] JSoup: Java HTML Parser. [Online]. Available: <http://jsoup.org>
- [4] Microsoft Bing API. [Online]. Available: <http://www.bing.com/developers/>
- [5] OpenEphyra Question Answering System. [Online]. Available: <http://www.ephyra.info/>
- [6] The Lemur Project. [Online]. Available: <http://www.lemurproject.org/>
- [7] Yahoo BOSS API. [Online]. Available: <http://developer.yahoo.com/search/boss/>
- [8] K. Balog, P. Serdyukov, and A.P. de Vries. Overview of the TREC 2010 Entity Track. In *Proceedings of TREC 2010*, 2010.
- [9] C. Chang and C. Lin. LIBSVM: A Library for Support Vector Machines. 2001. Available: <http://www.csie.ntu.edu.tw/~cjlin/libsvm>
- [10] Y. Fang, L. Si, N. Somasundaram, Z. Yu and Y. Xian. Purdue at TREC 2010 Entity Track. In *Proceedings of TREC 2010*, 2010.
- [11] J. R. Finkel, T. Grenager, and C. Manning. Incorporating Non-local Information into Information Extraction Systems by Gibbs Sampling. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics*, 2005.
- [12] J. J. Jiang and D. W. Conrath. Semantic Similarity Based on Corpus Statistics and Lexical Taxonomy. *Computing Research Repository – CORR*, 1997.
- [13] P. Jiang, Q. Yang, C. Zhang, and Z. Niu. Beijing Institute of Technology at TREC 2010: Notebook Paper. In *Proceedings of TREC 2010*, 2010.
- [14] D. Wang, Q. Wu, H. Chen, and J. Niu. A Multiple-Stage Framework for Related Entity Finding: FDWIM at TREC 2010 Entity Track. In *Proceedings of TREC 2010*, 2010.