NTNU at SemSearch 2011

Krisztian Balog, Marek Ciglan, Robert Neumayer, Wei Wei, Kjetil Nørvåg
Department of Computer and Information Science, Norwegian University of Science and Technology
Sem Sælands vei 7-9
Trondheim, Norway
{krisztib,ciglan,neumayer,wwei,noervaag}@idi.ntnu.no

ABSTRACT

In this paper we describe our participation in the Semantic Search Challenge of the Semantic Search 2011 Workshop. We focus on integrating multiple knowledge sources for the entity search task. With respect to the list search task we attempt to model human user behaviour when searching in Wikipedia. Since only preliminary results are available at the time of writing, we only present our approaches and do not draw any conclusions.

1. INTRODUCTION

The Database Systems Group of the Department of Computer and Information Science of the Norwegian University of Science and Technology (NTNU) participated in both tracks of the Semantic Search Challenge of the Semantic Search 2011 Workshop. Two tasks were investigated: *entity search*, where each query refers to one particular entity [3], and *list search*, where queries target a group of entities that match certain criteria (a task, similar in spirit to the List Completion problem at the INEX Entity Ranking track [2] and to the Entity List Completion task of the TREC Entity track [1]). The dataset for both tasks is the Billion Triple Challenge 2009 (BTC) collection.

Our main emphasis for the entity search task was on combining evidence from multiple knowledge sources, where each source is queried using a retrieval method tailored to its specific properties. For the list search task our goal was to mimic the behaviour of humans searching in Wikipedia for we believe much of the answers to list queries is available there, albeit not directly accessible. Finally, for both tasks, we exploited sameAs links extracted from DBPedia.

In the remainder of the paper, in two largely independent sections, we discuss our approaches to the entity search and list search tasks in Sections 2 and 3, respectively. We conclude and outline future directions in Section 4.

2. ENTITY SEARCH

In this section we outline our approach to the entity search track. This denotes to anwering queries that refer to one particular entity.

2.1 Retrieval model

We formulate the entity search problem as follows. We rank candidate entities (e) according to their probability of being relevant given the query q: $P(e|q)$. Instead of estimating this probability directly, we use Bayes' rule and rewrite it to:

$$P(e|q) = \frac{P(q|e) \cdot P(e)}{P(q)}. \quad (1)$$

Next, we drop the denominator as it does not influence the ranking of entities. The term $P(e)$ could be used to express the *a priori* belief that an entity is relevant (to any query); in this work, we assume this probability to be uniform. Hence, we rank entities according to $P(q|e)$.

To estimate $P(q|e)$ we consider a linear combination of three different entity representations: based on name only (N), based on DBPedia (D), and based on the BTC collection (B):

$$P(q|e) = \lambda_N P_N(q|e) + \lambda_D P_D(q|e) + (1 - \lambda_N - \lambda_D) P_B(q|e). \quad (2)$$

We take $P_N(q|e)$ to be either 0 or 1, based on strict string matching between the query and the name of the entity. $P_D(q|e)$, and $P_B(q|e)$ are estimated using a (fielded) Language Modeling approach. The mixture weights λ_N and λ_D were set to correspond with the importance of the individual sources. Intuitively, we assigned the following weights: $\lambda_N = 0.5$ and $\lambda_D = 0.3$. We detail the computation of these components in the next section.

2.2 Entity representations

2.2.1 Name-only representation

For each entity, we collected all its name variants from DBPedia (see Section 2.4.1 for the details). Let e_N denote the set of name variants that belongs to e. Based on this representation we make a binary decision:

$$P_N(q|e) = \begin{cases} 1, & \exists n \in e_N : match(n, q) \\ 0, & \text{otherwise,} \end{cases} \quad (3)$$

where $match(n, q)$ is a strict, case insensitive string matching function that returns true iff n equals to q.

2.2.2 DBPedia representation

We rank entities in DBPedia using a fielded Language Modeling (LM) approach. Each entity e is represented as a multinomial probability distribution over terms: e. The likelihood of the query given this model is then computed as a product of individual term probabilities:

$$P(q|\theta_e) = \prod_{t \in q} P(t|\theta_e)^{n(t,q)}, \quad (4)$$

WWW 2011, March 28–April 1, 2011, Hyderabad, India.
ACM 978-1-4503-0637-9/11/03.

where $n(t, q)$ denotes the number of times term t occurs in the query. So far, this approach equals to the standard LM approach. We deviate from it in the estimation of the entity language model θ_e:

$$P(t|\theta_e) = \sum_{f \in \mathcal{F}} P(t|\theta_{e_f}) \cdot P(f), \quad (5)$$

where $\mathcal{F}$ is the set of DBPedia fields considered, $P(t|\theta_{e_f})$ the term's probability given a specific field f, and $P(f)$ is the importance of that field. We estimate field-specific term probabilities as a linear combination of field-level and entity-level term probabilities, both smoothed by Dirichlet priors:

$$P(t|\theta_{e_f}) = (1 - \lambda_e) \cdot P(t|e_f) + \lambda_e \cdot P(t|e), \quad (6)$$

where

$$P(t|e_f) = \frac{n(t, e_f) + \mu_f \cdot P(t|\mathcal{C}_f)}{|e_f| + \mu_f} \quad (7)$$

and

$$P(t|e) = \frac{\sum_f n(t, e_f) + \mu \cdot P(t|\mathcal{C})}{\sum_f |e_f| + \mu}. \quad (8)$$

The components of Eq. 7 and Eq. 8 are as follows: $n(t, e_f)$ is the number of times term t appears in field f of entity e, $|e_f|$ is the length of field f of e (i.e., $\sum_t n(t, e_f)$), μ_f is a smoothing parameter for field f, μ is the entity-level smoothing parameter, $P(t|\mathcal{C}_f)$ is a field-specific background language model, and $P(t|\mathcal{C})$ is the general language model for the collection. The smoothing parameter μ_f was set to the average field length of f, i.e., $\sum_e |e_f|/|e|$, where e is the number of entities. Similarly, μ was set to the average entity representation length. The background models $P(t|\mathcal{C}_f)$ and $P(t|\mathcal{C})$ were calculated using a standard maximum-likelihood estimate on the corresponding representation. We set the λ_e parameter in Eq. 6 to a fixed value of 0.5. The fields and corresponding weights we used in are listed in Table 2.

2.2.3 BTC representation

We consider two representations based on the BTC collection. The first approach (`BTC singlefield`) renders triples as single-field documents, and ranks them using a standard LM approach. Specifically, we use Eq. 4 for ranking, where the entity model is estimated using Eq. 8. The other variation (`BTC name+content`) distinguishes between *name* and *content* fields. Using the retrieval model introduced in the previous subsection, we set $\mathcal{F} = \{\text{name}, \text{content}\}$ and consider the two fields equally important. λ_e (in Eq. 6) was set to 0.7 based on empirical results with last year's queries. Smoothing parameter estimation is as discussed before.

2.3 Exploiting sameAs relations

Additionally, we experimented with exploiting "sameAs" relations extracted from DBPedia. We propagate a fraction of the original query-likelihood scores along sameAs links:

$$\text{score}(q|e) = P(q|e) + \lambda_S \cdot \sum_{e' \in e_S} |P(q|e') - P(q|e)|, \quad (9)$$

where e_S denote the set of sameAs variants of entity e. We set λ_S to 0.75.

2.4 Preprocessing and Indexing

Both collections we used (DBPedia and BTC) were sorted prior to indexing to facilitate the indexing on a per-entity basis; every subject in the collections was treated as an entity, where the corresponding predicate-object values constitute to the entity's representation. For the indexing part, we used Apache Lucene[1]. Preprocessing was based on Lucene's standard analyzer, including lowercase transformation and basic stop word filtering. We processed all queries with the Yahoo! Spelling Suggestion API[2] and applied the same transformations (lowercasing and stop word removal) as to entity documents. Table 1 summarizes the indexes we built; next, we discuss the collection-specific details.

Index	#fields	#entities	#size
DBPedia	7	7.8M	20GB
BTC singlefield	1	23.9M	42GB
BTC name+content	2	38M	35GB

Table 1: Indexes used.

2.4.1 DBPedia

We used the most recent complete dump of DBPedia made available on the DBPedia homepage[3]. We performed additional preprocessing with respect to URI decoding and matching in order to be compatible with the BTC collection. Additionally, we filtered out those DBPedia URIs from the result list that do not exist in the BTC collection as a subject. To catch all dependencies we used reversed versions of the input files (with subject and object positions switched) for disambiguations, page links, and redirects. We did not perform exhaustive parameter tuning; we tried a few different configurations and used the one that performed best on the SemSearch 2010 queries. The list of fields used and their corresponding weights are shown in Table 2.

Field Name	Weight
short abstracts	0.10
long abstracts	0.10
article categories	0.07
disambiguations	0.20
infobox properties	0.11
labels	0.30
wikipedia links	0.12

Table 2: Fields in the DBPedia index.

We also used DBPedia to find name variants for exact name matches: for each DBPedia URI we considered the title and titles of pages redirecting to that URI as the set of name variations.

2.4.2 BTC

For the single field index (`BTC singlefield`) we ignored the predicate fields and concatenated all object fields that were literals. We filtered out subjects that had less than 50 characters of textual material associated with them. As to the multi-field index (`BTC name+content`) we manually identified predicates that hold names for the top 10 sources of the BTC collection. All other predicates were considered "content." Again, we limited ourselves to literal objects and filtered out subjects that had less than 40 characters worth of textual data in the content field.

[1] `http://lucene.apache.org/java/docs/index.html`
[2] `http://developer.yahoo.com/search/web/V1/spellingSuggestion.html`
[3] `http://wiki.dbpedia.org/Downloads36`

2.5 Submitted runs

Table 3 presents an overview of the runs we submitted. For each, we used the same approach to identifying exact name matches and to ranking entities in DBPedia.

RunID	BTC index	sameAs	MAP
NTNU-Olav	BTC singlefield	N	0.2072
NTNU-Harald	BTC name+content	N	0.2063
NTNU-Godfrid	BTC name+content	Y	0.2050

Table 3: Runs submitted to the entity search track.

3. LIST SEARCH

This section describes our approach to the list search task. We begin with a brief overview of our approach, followed by the description of the datasets we have used in Section 3.1. Finally, we present the procedure we used to generate answers for the input queries in Section 3.2.

Our approach to the list search task was inspired by the process that a human user would carry on to answer list queries were he asked to do so with the help of Wikipedia. This process would probably be to enter the query to the search field of the Wikipedia GUI and inspect the top k results, matching Wikipedia articles, for the correct answer. One could suspect that the items of the correct answer to the list query would be distinct Wikipedia articles themselves and would be linked from the top results of the full-text query issued against the index of Wikipedia articles. In this spirit, our approach relies on retrieving information from the index of long abstracts of Wikipedia articles; from the top k retrieved articles, we expand by hyperlinks to obtain the list of articles, representing candidate entities for the list query answer. We then check whether the candidate list contains Wikipedia article sets, if so, we boost the scores of members of these sets. Under the term Wikipedia article set, we understand a set of Wikipedia articles forming a semantic group; we describe Wikipedia sets used in this work in Section 3.1. If no sets are identified in the candidate list, we merely rely on boosting the score of the items from the candidate list related to the principal entity of the query.

3.1 List Search Datasets

This subsection provides a list and description of the data sets we have used for the list search task.

Wikipedia article index. Lucene index of the long abstracts of Wikipedia articles. More specifically, this index is used to retrieve the articles most related to the input query.

Wikipedia link graph. This data set contains the network of Wikipedia articles and links between them. An article is a node in the network and links correspond to the hyperlinks connecting articles. Each node has several attributes—Wikipedia identifier, article title and list of identifiers of the sets the article belongs to. The data set is used to form the candidate list from the top k most related articles to the query.

Wikipedia sets. This dataset contains sets of Wikipedia articles that form a semantically related group. We used the membership in Wikipedia categories and the inclusion of Wikipedia templates to generate the Wikipedia sets. For example, Wikipedia articles belonging to the category "Category:Astronauts" form one set in Wikipedia set dataset, and articles using the Wikipedia template "Template:Ancient_Cities_of_Cyprus" form another set (derived from the template inclusion in Wikipedia). We have also constructed an index of the Wikipedia sets. For each set a document was created by concatenating the short abstracts of articles belonging to that set. Those documents were indexed and we use the index to check the relevance of the input query to the sets identified as potential answers.

Annotation dictionary. This data set contains a mapping between strings and the Wikipedia articles referred to by those strings. We used the default dictionary used by Wikipedia miner[4], containing the mapping between anchor texts and articles, and extended it by adding article names and names of the redirect pages.

3.2 List Search Process

Here, we describe the main components of our approach to the list search task. It consists of the following steps (each of which will be described in more detail below): query analysis, querying the article abstracts index, generating the list of candidate items, boosting of scores of entities related to the main entity of the query, and, finally, boosting of scores of items belonging to Wikipedia sets.

1. Query analysis. Our first step is to analyze the query, which is facilitated by using the Wikipedia miner toolkit to annotate the query with Wikipedia topics. This gives us: a) query segmentation that we exploit in the full-text search step, b) entities (in form of Wikipedia article titles) that the query targets. We identify the principal entity of the query (the one with the highest relevance score from Wikipedia miner) and use it for query reformulation. If the query segment related to the principal entity is only slightly different from the entity article's title (the Levenshtein distance equals to 1), we try to reformulate the query by replacing the given query segment with the title of the entity article. We run both the original and the reformulated query against the index of Wikipedia abstracts. If the sum of the scores of top 10 items of the reformulated query is significantly higher (at least 2 times) than the sum of scores of the top 10 items from the original query, we use the reformulated query in the following steps. In the SemSearch Challenge dataset, the query reformulation was used instead of the original for two queries.

2. Querying the article abstracts index. We run two queries against the article abstracts index. The first one is evaluated by the Language Modeling approach described in 2.2. The second query is a boolean query with the following constraint: the terms from the text query segment related to the principal entity (identified in step 1) must be present in the target document. As the two result sets use different scoring functions, we merge the results by normalization; from each result set, we take the first 100 items, and normalize their scores (so that their sum is equal to 1), we then merge the results. Let T_k be the set of top k results of the merged ranks, and members of the set are Wikipedia articles. Let $r(i); i \in T_k$ be the rank of the item i in the merged result ranking.

3. Generating the list of candidate items. In this step, we take the top k (in the submitted runs $k = 10$) items from the previous step (T_k). We generate the list of candidate entities by taking those top k results and expanding from related articles by Wikipedia links. Every item added

[4] `http://wikipedia-miner.sourceforge.net/`

to the candidate list is assigned a score proportional to the rank of the item from T_k. If the item already exists in the candidate list the score is added to its existing score. In this way, the items referred by multiple links from the result set from the previous step will receive a higher score. More formally, let $G = V, E$ be the link graph of Wikipedia. Let $e(i,j) = 1 \Leftrightarrow e_{i,j} \in E$ and $e(i,j) = 0 \Leftrightarrow e_{i,j} \notin E$. We hence denote a candidate set C as:

$$C = \{j; j \in V \wedge \exists i \in T_k : e(i,j) = 1\}. \quad (10)$$

We set the score of item $c \in C$ to be:

$$w(c) = \sum_{i \ in T_k} e(i,c) \times (1 - ((r(i) - 1) \times (1/k))). \quad (11)$$

4. Boosting of items related to principal entity. We take the principal entity P identified in the query analysis (Step 1) and boost the scores of the entities in the candidate list that both are: a) linked to by the principal entity and b) link to the principal entity (in the link graph of Wikipedia). We boost the score of such an item I by computing the cosine similarity of the principal entity and the given item. In this case we represent Wikipedia articles as vectors constructed from their adjacency lists in the link graph. If $w(I)$ is the original score of the item, the boosted score $wb_1(I)$ is defined as follows: $wb_1(I) = w(I) \times sim(P,I) \times b_1$, where b_1 is the boost constant (in our experiments we used $b_1 = 100$) and $sim(P,I)$ is the cosine similarity of the adjacency vectors.

5. Boosting of scores of Wikipedia set items. In this step, we identify Wikipedia sets that have more than p fraction of their members in the candidate list. For each such set S we compute the similarity score of the query and the set document $D(S)$ (see Section 3.1), using the standard Lucene scoring function. We boost scores for the items from the candidate list according to:

$$wb_2(I) = w(I) \times b_2 \times \sum_{I \in S : |S \cap C| \geq p \times |S|} sim(q, D(S)). \quad (12)$$

Here, b_2 is the boost constant for sets, and $sim(q, D(S))$ is the similarity of the set document $D(S)$ and query q; in our runs we have used $p = 0.7$.

6. Postprocessing. In the postprocessing step, we sort the items in the candidate list in descending order according to their scores and we map results back to subjects in the BTC collection.

3.3 Submitted runs

We submitted three runs for the list search task. The first one followed the process as described in Section 3.2, only omitting Step 5.—boosting of the Wikipedia sets items. The second run exactly followed the approach presented in Section 3.2. The third run differed in the postprocessing step. For the items with a score higher than a defined threshold, have also added entities linked by sameAs relations.

3.4 Discussion

For the list search task, we limited our approach to the Wikipedia data set. The reason for this was purely pragmatic—while a part of the team was processing the BTC data set to a usable form, the rest of the team was experimenting with the list search task on the Wikipedia data set (which, in the form of DBPedia, is also covered in the BTC collection). Due to time constraints and the quality of pre-submission results we achieved, we decided to keep using only the Wikipedia dataset. However, our approach is fully transferable to the BTC collection or any other to RDF dataset in general, as we exploit textual descriptions of entities, links between them, and type information (in the form of categories/templates). One important observation from the results is that by exploiting Wikipedia sets and boosting sets' members we can achieve substantial improvements compared to our baseline. Considering sameAs variants of high scoring entities led to further performance improvements; see Table 4.

RunID	Wikipedia set boosts	sameAs	MAP
NTNU-1	N	N	0.1625
NTNU-2	Y	N	0.2594
NTNU-3	Y	Y	0.2790

Table 4: Runs submitted to the list search track.

4. CONCLUSIONS

In our participation we focused on integrating evidence from multiple sources for the entity search task: we employed a fielded Language Modeling approach to rank entities in the BTC collection and in DBPedia. Additionally, we considered strict name matches based on a dictionary of entity name variants extracted from DBPedia. As to the list search task we attempted to model human user behavior when searching in Wikipedia. Our approach includes a query analysis step to identify the principal entity in the query. In the ranking phase we utilize the Wikipedia link graph and semantically related article sets, defined by Wikipedia categories and templates.

Based on the preliminary results our best entity search run ranked third among all submissions and our list search runs were placed first, second, and third, of all runs. These initial results indicate that our approaches are very effective and show great promise for tackling these tasks.

In future work we plan to perform an exhaustive success and failure analysis based on the full system evaluations. As our approaches employ a number of parameters, most of which were set intuitively in the lack of time and—in case of the list search task—in the lack of training data, we believe that there is much to gain by adjusting these settings.

5. REFERENCES

[1] K. Balog, P. Serdyukov, and A. P. de Vries. Overview of the TREC 2010 entity track. In *Proceedings of the Nineteenth Text REtrieval Conference (TREC 2010)*. NIST, February 2011.

[2] G. Demartini, T. Iofciu, and A. P. De Vries. Overview of the INEX 2009 entity ranking track. In *Proceedings of the Focused retrieval and evaluation, and 8th international conference on Initiative for the evaluation of XML retrieval*, INEX'09, pages 254–264. Springer-Verlag, 2010.

[3] H. Halpin, D. M. Herzig, P. Mika, R. Blanco, J. Pound, H. S. Thompson, and D. T. Tran. Evaluating ad-hoc object retrieval. In *Proceedings of the International Workshop on Evaluation of Semantic Technologies (IWEST 2010)*, 2010.