Entity List Completion Using Set Expansion Techniques

Bhavana Dalvi, Jamie Callan, William Cohen
(bbd, callan+, wcohen)@cs.cmu.edu
Language Technologies Institute, CMU
Group Name : CMU_LIRA

February 28, 2011

Abstract

Set expansion refers to expanding a partial set of "seed" objects into a more complete set. In this paper, we focus on relation and list extraction techniques to perform Entity List Completion task through a two stage retrieval process. First stage takes given query_entity and target_entity examples as seeds and does set expansion. In second stage, only those candidates who have valid URI in Billion Triple dataset are ranked according to type match with given types. First stage of this system focuses on the recall while second stage tries to improve precision of the outputted list. We submitted the results on the Web as well as ClueWeb09 corpus.

1 Introduction

Entity List Completion task is a pilot task in TREC entity track, introduced this year to create a corpus for evaluation of semantic search systems. The motivation of this task is to complete a list of entities which have a particular relation with query entity. Also, they have to be located in the Semantic Web. For this year's task is based on Billion Triple Challenge 2009 dataset. Unlike main task in Entity Track where entities are represented by the homepage of the entity in Clueweb dataset, here they are represented using a Uniform Resource Identifier(URI) within the LOD. The task of Entity List Completion is defined as follow. Given a query entity, by its URI, the type of target entity, nature of their relation, as well as few examples of target entities, find remaining entities that are related to query entity, of target type, which are part of LOD.

In TREC 2009, many successful systems used DBpedia and Freebase ontologies to answer entity track queries. So we tried using the knowledge base developed at CMU by Read The Web group [5]. Read The Web group used coupled semi-supervised learning for doing ontology driven information extraction [4]. The knowledge base currently contains 440K beliefs and most of them are popular facts frequently found on the Web. When we tried to answer Entity Finding Task queries with this knowledge base, coverage was low. This might be because the query entities are less popular ones in the Web corpus. We also worked on a corpus of noun-phrase pairs data extracted by shallow parsing the

	Country names	Reality TV shows
Input seeds	India China Canada	Amazing Race Survivor ...
Output	Japan France Brazil Australia Germany	Big Brother The Mole The Apprentice Project Runway ...

Table 1: SEAL input/output examples

valid sentences from Clueweb dataset [2]. We found that coverage of queries was still low on this data. This was because these noun-pairs were capturing the occurrences of noun-phrases which occur in sentences separated by context string. The queries in this track are such that most of the answer entities occur on homepage of query entity or some page linked to homepage. Hence any approach which tries to find the query entity and target entity separated by a context will fail.

In this paper we apply set expansion technique for the Entity List completion task. This technique is based on the assumption that the target entities occur together in structured webpages, and have similar character patterns around them. It then leverages such co-occurring patterns to suggest entities similar to given set of seed entities. To solve the Entity List Completion task, we proposed a two stage approach. First stage uses the list extraction technique called "Set Expansion for Any Language (SEAL)" which takes given query entity and examples of target entities as seeds and generates a list of candidate entities to complete the list. In the second stage, we rank these entities according to type match with given allowed target types. Section 2 describes SEAL system. Section 3 describes our two stage retrieval process. Submitted runs are discussed in Section 4. Section 5 gives conclusion and Future work.

2 Set Expansion for Any Language (SEAL)

Set expansion refers to expanding a partial set of "seed" objects into a more complete set. Table 1 shows the examples of SEAL's input/output. e.g. given two names of Reality TV shows, SEAL finds other reality TV shows.

The SEAL system [6] has three components: the Fetcher, the Extractor, and the Ranker. The Fetcher fetches web pages from the World Wide Web. The URLs to crawl are decided based on top results provided by the Search engine, when queried for concatenation of seeds. Given the seed entities and the webpages on which they co-occur, the Extractor learns wrappers(regular expressions surrounding the seed entities) for each page, and then applies those wrappers, to extract more candidate entities that can be part of list of seeds. The last stage of set expansion process, comprises of the Ranker which builds a graph, and then ranks the extracted mentions globally based on the weights computed in the graph walk.

Figure 1 describes the above process pictorially. This system is modular,

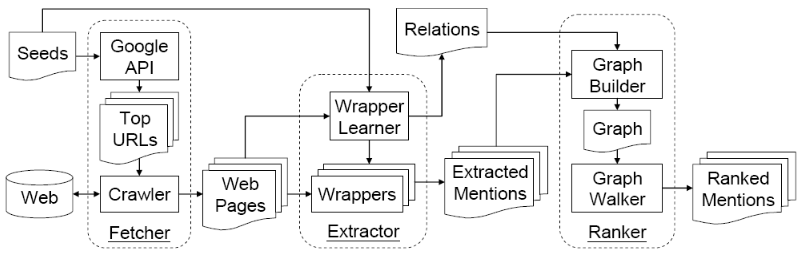

Figure 1: Flow chart of SEAL System

hence Google API in the Fetcher component of the system can be replaced by any search engine API by creating appropriate interface for it. Similarly in the Ranker module, Graph walk based ranker can be replaced by any other ranking method. We describe the various configurations used for TREC 2010 experiments in Section 3.

SEAL has already shown promising results when used in conjunction with Ephyra System for the task of list question answering task (TREC 2006-07) [8]. Iterative SEAL [7] is an extension of the SEAL system which enables us to provide many seeds as input, and system iteratively chooses small number of them at a time and does set expansion. It also enables to feed the high score candidates at the end of iteration 1 to act as seeds for iteration 2, leading to a bootstrapping process.

SEAL defines a wrapper as a pair of maximally-long left and right context strings (l; r) that bracket at least one occurrence of every seed on a web page. Since longer strings are better, they have defined a ranking algorithm called Wrapper Length which works as follows :

$score(x) = \prod_{j\ extracts\ x}(length(w_j))$
where w_j is the jth wrapper composed of a pair of left and right contextual strings, and the function length returns the sum of the character lengths of those pair of strings in w_j. This heuristic is based on the assumption that an item should have a high score if it is extracted by many long wrappers. This simple ranking function worked very well with iterative SEAL [7]. The same algorithm can be extended for extracting binary relations. To do this task we need a third type of context, called the middle context that occurs between the left and right contexts of a wrapper for separating any two items. The same algorithm as before can be applied, except that a seed instance in the algorithm is now a seed instance pair bracketing some middle context (i.e. "s1 <middle-context> s2 ").

3 Two Stage Retrieval Process

In this section we describe in detail our two stage retrieval process for the ELC task. In the first stage we use the SEAL system for set expansion. SEAL is capable of handling relation instances. So we first feed {query_entity, target_entity}

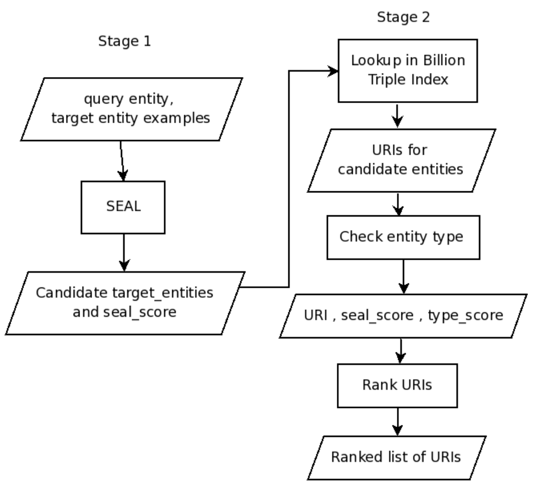
Stage 1
query entity, target entity examples
SEAL
Candidate target_entities and seal_score
Stage 2
Lookup in Billion Triple Index
URIs for candidate entities
Check entity type
URI , seal_score , type_score
Rank URIs
Ranked list of URIs

Figure 2: Flow chart of our two-stage System

pairs to SEAL. SEAL learns wrappers around these relation instances and generates a candidate list of more relation instances i.e. entity-pairs. We filter out those entity-pairs which have query_entity as part of them. The other half of the pair becomes candidate answer for ELC task. If SEAL is unable to find relation instances, we use only target_entity examples as seeds. SEAL then returns the entities which can fit into the given list, which also become candidate answer for ELC task. SEAL outputs score along with each candidate it outputs. At the end of stage 1, we have a list of candidate {entity, SEAL_score} pairs.

Second stage deals with type-checking and ranking. We used the Billion triple index [1]. For each entity in the candidate list produced by first stage, we do a lookup in Billion Triple Index. For each query, the index returns multiple URIs. We extracted the type fields in the webpages corresponding to top URIs. ELC task specifies two fields target_type_dbpedia and target_entity. For many queries target_type_dbpedia indicates more specific type (e.g. Scientist) than target_entity (e.g. Person). We first try to match target_type_dbpedia, if not then target_entity. This step assigns type_score to candidate URI. A type_score of 2 is assigned for target_type_dbpedia match, 1 for target_entity match and 0 for no type_match but a valid URI. Candidate entities for which no URI was found are removed from the set. At this stage we have candidate tuples {entity_name, URI, type_score, SEAL_score}. We rank the entities first by type_score and then resolve the collisions using SEAL_score. These entities are then outputted in the TREC result format.

4 Experiments and Submitted Runs

SEAL has an interface for querying the Google API to collect the documents in which seeds co-occur. We implemented another interface to query the Clueweb index [3]. We presented two runs for the ELC task. Run-1 was using the Google API and Run-2 was using Clueweb API. Since Clueweb is a subset of actual web, SEAL could not answer 2 queries which were answered using Google API. Table

	Run-1	Run-2
#Queries answered (Total 14)	11	9
#Queries answered by relation extraction	4	4
#Queries answered by list extraction	7	5
#URIs that matched target_type_dbpedia	412	406
#URIs that matched target_entity	56	64

Table 2: Overall statistics about two runs

Queryid	1	2	3	4	5	6	7	11	12	15	16	17	19	20
Run-1	4	0	0	6	0	19	81	31	95	100	0	1	39	92
Run-2	4	0	0	6	0	97	22	0	96	58	0	1	94	92

Table 3: Number of target entities outputted by each run per query

Run	MAP	R-precision
Run-1	0.0228	0.0274
Run-2	0.0755	0.0494

Table 4: Overall performance of the two runs

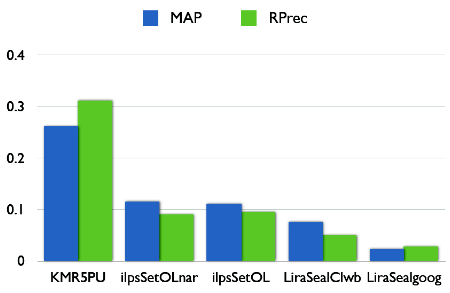

Figure 3: Performance comparison of all participating runs

2 gives the detailed statistics about how many queries were answered in each run and number of URIs that matched the given target type. Table 3 describes the number of target entities the system generated per query in each of the runs. Table 4 shows the overall performance of both runs using two metrics, Mean Average Precision(MAP) and R-precision.

When we manually checked the answers and evaluation done for each query, we found that our method fails to find correct URIs for many result entities. E.g. for query-4 ("Professional sports teams in Philadelphia") our system could find the sports team names : Soul, Phantoms, Phillies, Eagles etc. but it does not return correct URIs. We need more sophisticated URI finding algorithm to fix this problem. In queries like query-17 ("Chefs with a show on the Food Network"), where SEAL relation extraction failed, and we used SEAL list extraction, we got the target entities which were not relevant to the query entity as per the given description.

Figure 3 shows the performance comparison of all the participating runs for this task. Our system did not not perform well when compared to other participants. This evaluation helped us understand two improvements that can be done in our system : (1) Improve relation extraction method by using query description and (2) Devise better method for finding entity URI. We believe that these two changes will improve MAP and R-precision of our system.

5 Conclusions and Future Work

We developed a two stage retrieval process to answer entity list completion queries. First stage focused on improving recall by doing set expansion on query entity and examples of target entity. Second stage improves precision by checking the type of candidates found in first stage and ranking the results by type_score and a SEAL generated score. Our system needs better relation extraction and URI finding methods to perform reasonably well on this task. In future we are want to focus on three aspects of the system : 1) Devise better method of finding the correct URI, given name of a target-entity. 2) Extend SEAL to make use of the relation description given in the query. If our system can learn wrappers taking into account the kind of relation we want to learn then it will definitely improve accuracy of list extraction. 3) Study the relation of entities appearing on homepage of the query_entity and how to define/extract such relations.

Acknowledgments
This work was supported by the Google Research Grant. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors.

References

[1] K. Balog. Billion triple index. `http://zookst1.science.uva.nl:8888/btc-webapp-0.1/btc2009`.

[2] J. Callan. The clueweb09 dataset. `http://boston.lti.cs.cmu.edu/Data/clueweb09/`.

[3] J. Callan. Search clueweb09 category a - english. `http://boston.lti.cs.cmu.edu:8085/clueweb09/search/cata_english/lemur.cgi`.

[4] A. Carlson, J. Betteridge, R. C. Wang, E. R. Hruschka, Jr., and T. M. Mitchell. Coupled semi-supervised learning for information extraction. In *WSDM*, 2010.

[5] T. Mitchell. Read the web project. `http://rtw.ml.cmu.edu/rtw/`.

[6] R. C. Wang and W. W. Cohen. Language-independent set expansion of named entities using the web. In *ICDM*, 2007.

[7] R. C. Wang and W. W. Cohen. Iterative set expansion of named entities using the web. In *ICDM*, 2008.

[8] R. C. Wang, N. Schlaefer, W. W. Cohen, and E. Nyberg. Automatic set expansion for list question answering. In *EMNLP*, 2008.