A Pattern Matching Approach to Streaming Slot Filling

Hung Nguyen, Yi Fang, Sandhya Gade, Vijay Mysore, Juan Hu, Sabita Pandit,
Aparna Srinivasan, Miao Jiang

Department of Computer Engineering, Santa Clara University
500 El Camino Real, Santa Clara, CA 95053, USA

Abstract

In this paper, we described our system for Knowledge Base Acceleration (KBA) Track at TREC 2013. The KBA Track has two tasks, CCR and SSF. Our approach consists of two major steps: selecting documents and extracting slot values. Selecting documents is to look for and save the documents that mention the entities of interest. The second step involves with generating seed patterns to extract the slot values and computing confidence score.

1. Introduction

KBA is the new track in NIST and was started in 2012. In its first year, KBA had one only task, Cumulative Citation Recommendation (CCR.) In 2013, Streaming Slot Filling (SSF) task was added into KBA track. KBA systems aim at helping large knowledge bases such as Wikipedia to automatically filter streams for documents relevant to a set of entities. The size of the streaming corpus this year is much larger than that in 2012. The corpus contains data from 10/05/2012 to 02/13/2013. Each day has 24 time slot, running from 00 to 23. The data includes web log, news, social, and classified text streaming on the internet. There are total 538,359,206 documents in the corpus. We have total of 170 targeted entities where 150 of the entities came from Wikipedia, and 20 of the entities came from Twitter. Each entity is one of these three types: Facility (FAC), Person (PER), Organization (ORG).

2. Methodology

Our system consists of the following steps.

- Accessing data in corpus
  - Decrypting with gpg
  - Unzipping
  - De-serializing with Thrift library
- Selecting documents
- Removing duplicated documents
- Generating seed patterns
- Retrieving relevant sentences, using Indri
- Tagging relevant sentences, using Stanford NLP
- Pattern matching

The basis of our system is using seed patterns to extract the value of every slot name for each entity. For each slot name, we came up with a list of seed patterns. For example, with an entity of type PERSON, a slot name "FoundOf", assuming that person is the founder of company C. Then we expect to find in documents the following sentences: "E started C", "E founded C", "C was founded by E"…, then our seed patterns would be "E started", "E founded", and "was founded by E". Then assuming we found a sentence that contains the seed pattern, for example "E started K in 2012." So we will check the type of the token K. If its type is "ORG", then we can conclude that the PERSON E is the founder of the ORGANIZATION K. And we will update the slot "FoundOf" of the entity E. The slot value would be K.

Pattern Matching Method

3. Document filtering

It is not scalable to search and extract slot values for every entity in each of the document [4]. To reduce processing time, we need to filter relevant documents in the corpus and download to local system only the documents that are relevant to at least one entity.

We ran document selection on Amazon EC2 Linux instances[1]. The process of document selection went through the following steps

a. Accessing the text data in the corpus
   - Decrypt the files using pgp utility
   - Unzip the files with zx program
   - Decoding document using Thrift library. The Thrift library supports many languages. To achieve the highest performance, we use Thrift for C++.

b. Select documents

   Once we have access to text in documents in the corpus, we will need to decide if the document is relevant by searching for exact match of the entity name. If we find the match, we will download the document to our local server. When download the documents, we group them by timestamp for further processing. It was stated that 4% of the Wikipedia citations did not mention the Wikipedia entities [1]. To cover this case, we use WordNet[2] to find synonyms of the entity and search for the synonyms. We perform on plain text instead of html version of data.

[1] http://aws.amazon.com/ec2/
[2] http://wordnet.princeton.edu/

c. Remove duplicated documents

In this step, we scan through all documents and read document IDs. We maintain the document ID list. For each document ID, we check in the list, if the document ID does not exist, we add to the list, and move to the next document. If the document ID exists, we delete this document because it is duplicated. Because we group documents by timestamp in step 2, when removing duplicated documents, the document with the earliest timestamp will be kept.

4. Slot Value Retrieval

a. Finding passages or sentences that contain information of the entities.

After finding documents and removing duplicates, we need to extract the sentences or the passages that contain the target entities in order to detect slot values of the entities. We used the Indri query language [5] to perform phrase matching. We needed to perform the search for each pair <e, d> where e is the entity name and d is the document. In our run, we used the formula #od20(…) to perform the search of the entity name in the document. The meaning of #od20 is to request the Indri query language to search the text in ordered window. The term must appear in order, with at most 19 words between each the terms in parentheses. If the Indri language query finds the sentences or passages that satisfy the condition, Indri will output the relevant sentences or passages. For each of the relevant sentences output from Indri, we will need to keep track the document ID and entity name related to these sentences.

This fragment of the xml file shows the way we format the input file to feed into the Indri language query.

```
<query> find
  <number>1</number>
  <text>#combine[sentence](#od20(Stuart Powell Field))</text>
</query>
```

The reason we use od20 to perform Indri language query is to handle different variances of the seed patterns. For example, consider the seed pattern “E founded C”. The following sentences are example of variances of this seed pattern:

- E, graduated in 2000, founded C
- E, a former student of MIT, founded C
- E and Mark founded C
- E, in 10/2000, founded C

If we used exact match, i.e. od1, we would only find the sentences that contains “E founded C”. When using od20, we would find sentences that contain any of the variances above.

b. Retrieving slot values by pattern matching

One essential condition we needed was the correct type for each token in the sentences. The documents in the corpus did not have that information for us. To solve this problem, we used Stanford Named Entity Recognizer (NER) library[3]. The Stanford NLP will tag the tokens in the input sentences.

With the passages or sentences that are associated with the entity and the seed pattern, we will be able to find the slot value. In the three seed patterns example above, "E started", "E founded", and "was founded by E", the first two seed patterns will have the slot value on their right, and the third pattern will have the slot value on its left. Another factor to help us detect the slot value is the type of the token. In this example, the token we are searching must have the type of ORG.

The process above explained how we detect the slot value in general. However, there are some exceptions that required us to search for slot values with different methods. First, for the slot name of "AssociatedOf", there is not a clear definition of what type of association. It is hard to define seed patterns. What we did for this case is searching another token of type PER or ORG in the same sentence with the entity we are interested in. If we found one, that person or organization will be considered associated with the entity. The second case is the slot name of "Title". We built an exhausted list of titles from various knowledge bases such as Wikipedia. For each entity E, we built the seed patterns by combine the title with the entity name. For instance, we would have seed patterns such as "Professor E", "Dr. E", "President E"… and so on. We would then search for exact match to find any seed patterns in the document. If we found "Professor E" in the document, we can conclude the title of entity E is "Professor".

5. Confidence Scores

One of the requirements in SSF run submissions is confidence score. The confidence score must be normalized to the range (1, 1000]. The confidence score needs to be generated by our system for each of the slot value that was retrieved from documents. The higher the score, the slot value is more likely to be correct. These scores represent the confidence level of the retrieved values.

In general, the confidence score is calculated as follows. Assuming we are retrieving information for entity E, slot name T, in document D. We might find zero, one, or multiple occurrences of value V for slot T of entity E. For the case of zero value found, we do not need to have confidence score. If we found only one value V, the score will be 1000. If we found multiple different values V, we will scale the score for each value as follows.

$$ConfidenceScore(V_i) = \frac{(c_i - c_{min}) * 1000}{(c_{max} - c_{min})}$$

[3] http://nlp.stanford.edu/downloads/CRF-NER.shtml

In this formula c_i is the repetition of value V_i, c_{min} is the minimum value of (c_1, c_2, … c_n) where n is the number of different values V we retrieve in document D, and c_{max} if the maximum value of (c_1, c_2, … c_n.)

In some cases, the slot name can accept multiple values. For example, a person may have many titles, such as "President", "Professor", … at the same time. The above confidence score formula is not applicable to this case. We adjusted the confidence score to 1000 for slot name "Titles" if we detect the title of the entity. For the purpose of experimenting, we submitted the runs for both before and after adjusting confidence score (see the experimental results)

6. Relevant Ratings

Along with confidence score for each slot value, we are required to have relevant rating. Relevant rating can have the following values

- -1, Garbage

Slot value has no information about targeted entity.

- 0, Neutral

Slot value is informative, but not citable.

- 1, Useful

Slot value is possibly citable, but not timely.

- 2, Vital

Slot value has timely information of the entity's current state, actions, or situation.

We assign the relevant rating for each value based on its confidence score as follows.

- [800, 1000] => relevant rating = 2
- [550, 800) => relevant rating = 1
- [250, 550) => relevant rating = 0
- (0, 250) => relevant rating = -1

7. Preliminary Results

As showed in the last year's CCR task, the cut-off confidence score will affect the precision of the results [2]. Figure 1 (a) shows our result before adjusting Confidence Score, and Figure 1 (b) shows the result after adjusting confidence score. As we can see, we have a better result after we adjust the confidence score.

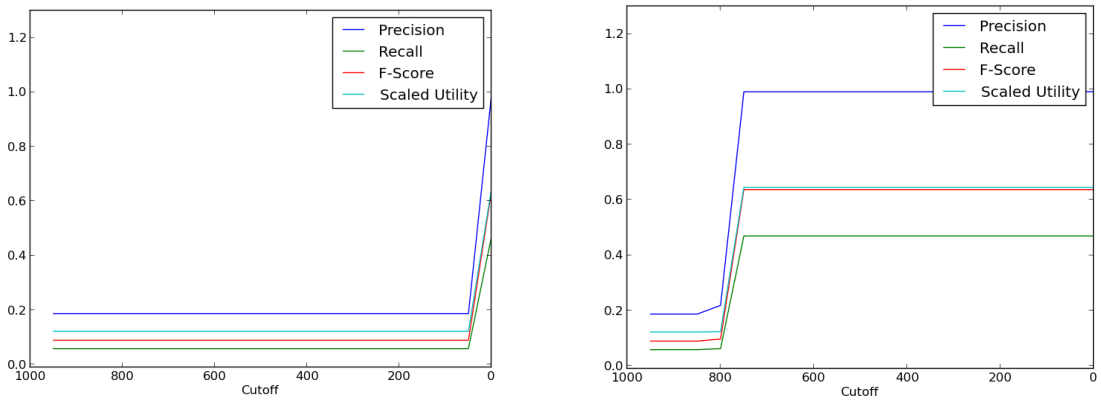

Figure 1. DATE-HOUR, All Entities, All Slots, (a) Before Adjusting Confidence Score DATE-HOUR; (b) After Adjusting Confidence Score

8. Future Work

Due to the large amount of documents, the selecting documents process took more than a month to finish, although we ran on multiple Amazon EC2 instances. The reason was that we need to loop through each document once for each entity. We will need to find a better algorithm to improve the performance and reduce the running time of Selecting Documents process. Our goal is to avoid looping through the document once for each target entity.

We also see that our confidence scores are sometimes not very reasonable. We will need to incorporate the time information of the document into the confidence score calculation. We also might need to consider having different formulas to calculate confidence scores for different slot types.

References

[1] S. Araujo, G. Gebremeskel, J. He, C. Bosscarino, and A. de Vries. CWI at TREC 2012, KBA track and Session Track, TREC, 2012.

[2] B. Kjersten, and P. McNamee. The HLTCOE Approach to the TREC 2012 KBA Track, TREC, 2012.

[3] L. Bonnefoy, V Bouvier, and P. Bellot. LSIS/LIA at TREC 2012 Knowledge Base Acceleration, TREC, 2012.

[4] K. Balog, H. Ramampiaro, N. Takhirov, and K. Norvag. Multi-step Classification Approaches to Cumulative Citation Recommendation. OAIR'13, May 22-24, 2013.

[5] T. Strohman, D. Metzler, and H. Turtle, and B. Croft. Indri: A Language Model-based Search Engine for Complex Queries. In Proceedings of the International conference on Intelligence Analysis, 2004.