Entity Profile based Approach in Automatic Knowledge Finding

Xitong Liu, Hui Fang
University of Delaware
Newark, DE, USA
{xtliu,hfang}@udel.edu

1 Introduction

We focus on the problem of profile building in this year's KBA track and proposed two methods. The first method is a baseline, which selects the stream items that has exact string match with the query entity. All the matched documents are assigned with the same relevance score. In the second method, we propose to use the related entities to help us identify the information related to the query entity. In particular, we retrieve the wikipedia pages for the query entities and extract the anchor text in all the internal links within the wikipedia page. These anchor text are treated as the related entities of the query entity and they are used to build the profile of the query entity. Given a stream item (i.e., a document), the relevance score is estimated by integrating the match with the query entity and the match with the related entities. Results on the training data show that the second method is more effective.

2 Entity Profile Building

The input query set for the KBA system is a list of 29 entities from English collection of Wikipedia. All the entities are manually selected by the KBA organizers and most of the entities are celebrities and selected from the Living_people category. A few of them are organizations. Moreover, the organizers "focused on entities with complex link graphs of relationships with other active entities". Therefore, it means each entity has rich link relation with other entities in the Wikipedia, which makes it possible for us to exploit such relations to build the entity profiles.

To solve the first challenge, i.e., entity profile building, we propose a general approach by collecting the internal links with the Wikipedia page of each query entity e_q and use the anchor text of the internal links as related entities of e_q.

Given an entity, the first thing is to retrieve its Wikipedia page. Fortunately, as the query set were selected from the Wikipedia collection directly, each entity are defined by its so-called *urlname*. The URL of the entitys Wikipedia page can be constructed by just appending the entity name to a Wikipedia base URL (i.e., `http://en.wikipedia.org/wiki/`). For instance, one of the 29 query entities is "Basic_Element_(music_group)", we can get its authentic Wikipedia URL as "`http://en.wikipedia.org/wiki/Basic_Element_(music_group)`". Instead of retrieving the HTML Wikipedia page directly, we utilize the API provided by Wikipedia to dump the raw content in `json` format. Given a query entity e_q, the API accepts *urlname* as input and returns the English Wikipedia page $wiki(e_q)$ in Wiki markup[1] format. The reason to parse Wiki markup instead of HTML is that we think it is much easier to identify and extract the internal links.

[1]Wiki markup: http://en.wikipedia.org/wiki/Help:Wiki_markup

With the retrieved Wikipedia page, we then apply a python based parser to parse the Wiki markup document and extract the related entities. Basically there are two types of links for a Wikipedia page: internal link and external link. The former connects the entities within Wikipedia and the the latter provides supplemental information with regard to the entity. An internal link between two entities indicates there are some relation between them. Therefore by following the internal links from the query entity e_q, we can get a list of entities which can be treated as related entities. More specifically, the internal link in Wiki markup document would be denoted as shown in the following example:

```
'''Basic Element''' is a [[Sweden|Swedish]] [[eurodance]]  [[hip-hop]] group formed
in 1993.
```

There are three internal links in the example above, each of which is embraced by double square bracket. The first link `[[Sweden|Swedish]]` contains two parts, separated by a vertical bar. The first part is the *urlname*, and the second part is the anchor text of the link shown on the rendered HTML page. Therefore, this link would be rendered as a link to `http://en.wikipedia.org/wiki/Sweden` with the anchor text "Swedish". The following two links `[[eurodance]] [[hip-hop]]` just have one part, and the text within the double square bracket serves as both the *urlname* and the anchor text. Therefore, we can extract three related entities from the markup text above, i.e., `Sweden`, `eurodance` and `hip-hop`. Formally, given a Wikipedia page $wiki(e_q)$, we can extract a set of related entities $rel(e_q) = \{e|e \in E(wiki(e_q))\}$ where $E(wiki(e_q))$ denotes all the Wikipedia entities in $wiki(e_q)$.

With the set of related entities, we define the entity profile $profile(e_q)$ as follows:

$$profile(e_q) = \{e_q, rel(e_q)\}. \tag{1}$$

3 Entity Profile based Stream Filtering

We now discuss how to use the entity profile $profile(e_q)$ to do stream filtering. Given a stream document d, we need to estimate how likely the document is relevant to the query entity e_q. As shown in Equation (1), an entity profile $profile(e_q)$ consists of the entity itself e_q and its related entities $rel(e_q)$, therefore, the relevance of the document should be determined by both e_q and $rel(e_q)$. To reflect this, we propose the following method to estimate the relevance between d and $profile(e_q)$:

$$score(d, e_q) = \alpha \cdot mention(d, e_q) + \beta \cdot \sum_{e \in rel(e_q)} occ(d, e), \tag{2}$$

where $mention(d, e_q)$ is an function which identifies the document d mentions e_q and it is defined as:

$$mention(d, e_q) = \begin{cases} 1 & \text{if } d \text{ mentions } e_q, \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

Moreover, $occ(d, e)$ denotes the occurrences of e in d. α and β are the coefficients which assign different weights to different score components to balance their influences. The main idea behind Equation (2) is that we want to capture whether the document mentions e_q as well as any related entities in $rel(e_q)$. The first component ($\alpha \cdot mention(d, e_q)$) checks whether e_q is discussed in d, and the second component ($\beta \cdot \sum_{e \in rel(e_q)} occ(d, e)$) serves as the complementary information to the relevance score under the assumption that the more related entities occur in d, the more likely d is relevant to e_q. Since the first component is the main body of the relevance score, α should be much larger than β to reflect it.

With the relevance score calculated, we then set a threshold $\mathcal{T}$ to determine whether the document is relevant to the query topic e_q or not. The stream document with the relevance score above $\mathcal{T}$ will be kept and others will be discarded.

Evaluation Set	Central		Central+Relevant	
Run	maxF	maxSU	maxF	maxSU
all-mean	0.220	0.311	0.404	0.498
all-median	0.289	0.333	0.553	0.554
UDInfo-KBA_EX	0.297	0.154	0.605	0.580
UDInfo-KBA_WIKI1	0.342	**0.331**	**0.639**	0.611
UDInfo-KBA_WIKI2	0.354	**0.331**	0.636	**0.617**
UDInfo-KBA_WIKI3	**0.355**	**0.331**	0.597	0.592

Table 1: Results of official runs. The maxF and maxSU measures are reported by the official evaluation program, which collects the maximum F1 and SU for each topic at certain relevance score cutoff and report the average then. **all-mean** and **all-median** are the mean and median of results aggregated from all the submitted runs in this year's KBA track respectively.

4 Experiment Results

4.1 Submitted Runs

We submitted four official runs to the KBA track. The main difference between them are α, β in Equation (2) and filtering threshold $\mathcal{T}$. The detail description are summarized as follows.

1. **UDInfo-KBA_EX**: $\alpha = 1000$, $\beta = 0$ and $\mathcal{T} = 0$. It is a special case and Equation (2) falls back to $score(d, e_q) = 1000 \cdot mention(d, e_q)$, which means the relevance score is estimated based on whether there is an *exact match* of query entity e_q in document d. If there is an exact match, the relevance score would be 1000. Otherwise, it would be 0. This run serves as a baseline.

2. **UDInfo-KBA_WIKI1**: $\alpha = 100$, $\beta = 1$ and $\mathcal{T} = 101$. The main idea of this method is that besides the exact match, we also want to capture the match of the related entities. $\mathcal{T}$ is set empirically based on the results of training data.

3. **UDInfo-KBA_WIKI2**: $\alpha = 100$, $\beta = 1$ and $\mathcal{T} = 102$.

4. **UDInfo-KBA_WIKI3**: $\alpha = 100$, $\beta = 1$ and $\mathcal{T} = 103$.

4.2 Results Analysis

The results of all the runs are summarized in Table 1. We can find that all of our four runs can reach good results among all the submitted runs. Moreover, by incorporating the match of the related entities into the estimation of relevance score, the performance can be improved, showing the effectiveness of related entities in the entity profile based filtering.

To better understand the performance of each query, we plot the maxF per query on both *central* and *central+relevant* to compare them side by side, as shown in Figure 1 and Figure 2, respectively. We can find that generally our entity profile based filtering method can outperform both the baseline and **all-mean** on most queries.

5 Conclusion

It is the first year of KBA track, and the task is relatively simple as the organizers want to get first impression on how the data would fit the task of knowledge base acceleration. We propose an entity profile based filtering framework and derive two methods to solve this year's task. Experiment results on the testing data show that our methods are effective to select the relevant documents. We find that by incorporating the related entities into the entity profile can improve the performance, showing that the related entities are important on finding the relevant documents.

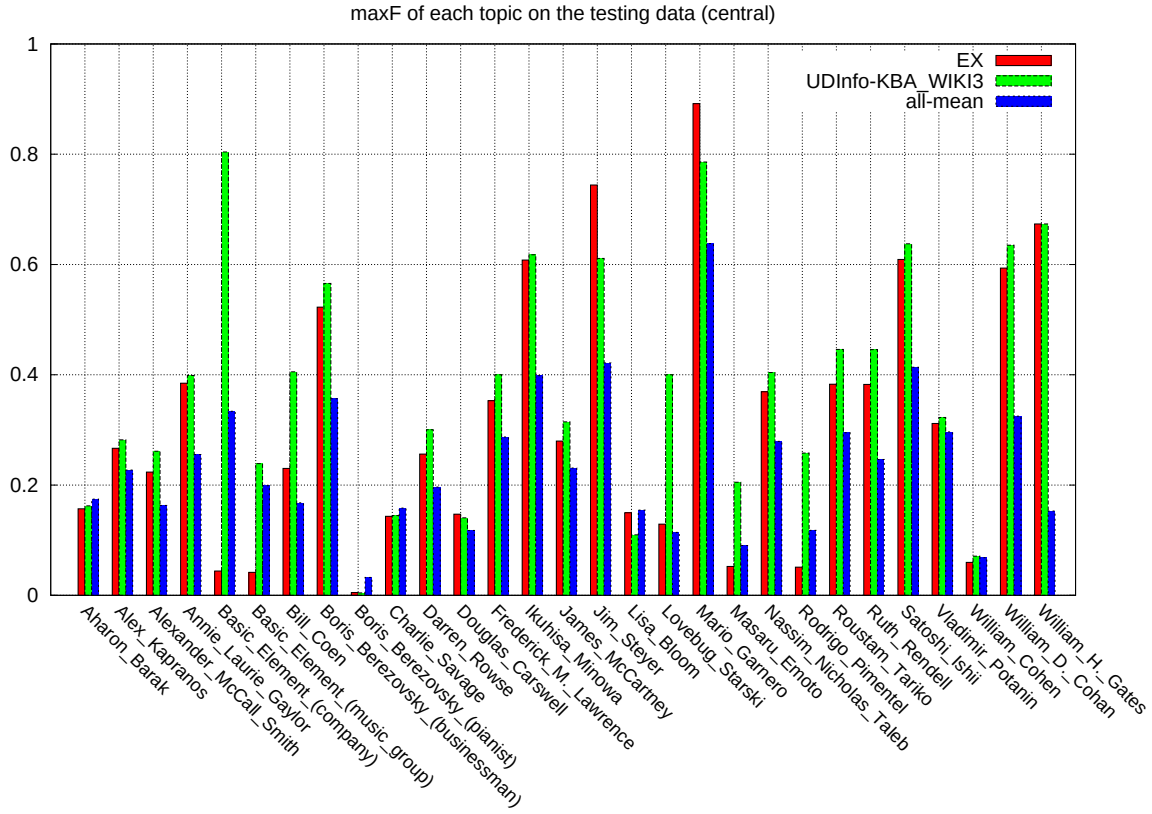

Figure 1: maxF of each topic on the testing data (central).

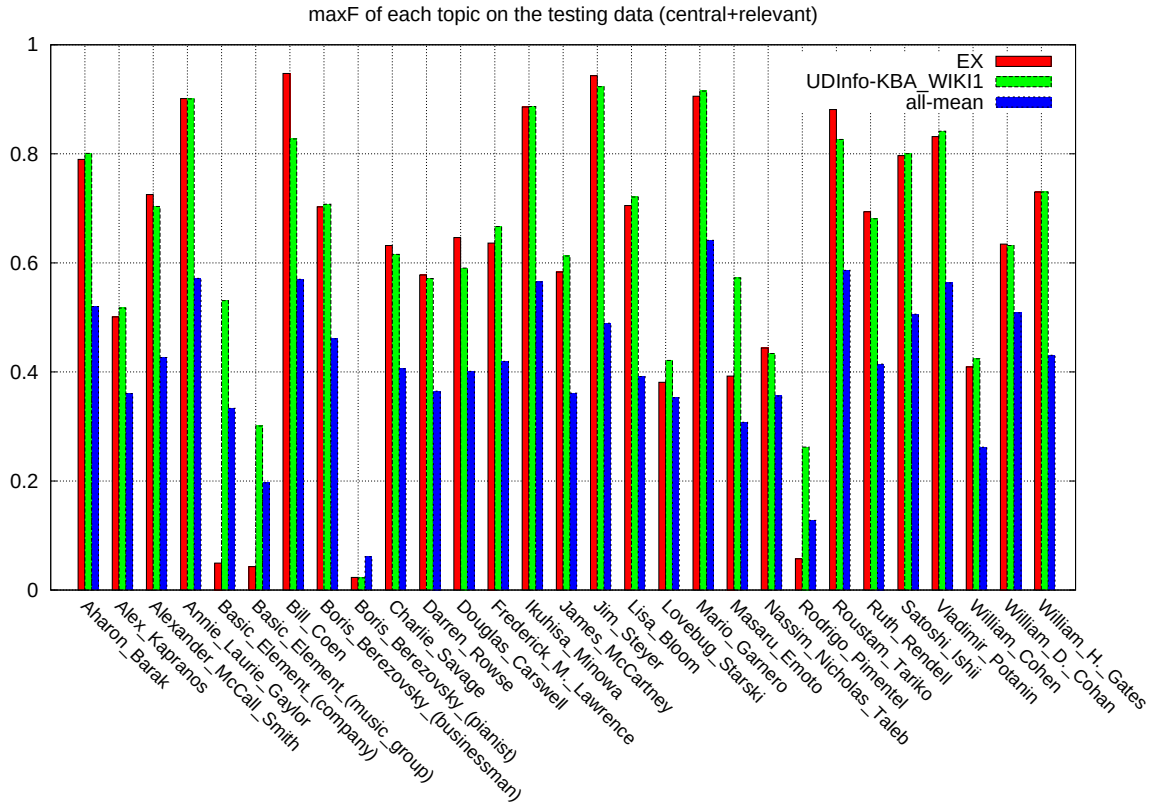

Figure 2: maxF of each topic on the testing data (central+relevant).