University of Waterloo at TREC 2014 Contextual Suggestion: Experiments with suggestion clustering

Luchen Tan Adriel Dean-Hall Pragnya Addala Charles L.A. Clarke
David R. Cheriton School of Computer Science, University of Waterloo
{luchen.tan, adeanhal, paddala}@uwaterloo.ca, claclark@plg.uwaterloo.ca

1 Introduction

In this work we present our group's first attempt at developing a system to solve the problem presented in the contextual suggestion task. As part of TREC 2014 the contextual suggestion track is running for the third time. The goal of this task is to tailor point-of-interest suggestions to users according to this preferences. Here we present how we gathered candidate points-of-interest, grouped them according to similarity using clustering, and picked points-of-interest that each user would find especially appealing.

The organizers of this track distributed users' personal profiles in three files: examples2014.csv, profiles2014-70.csv and profiles2014-100.csv. A list of 100 example points-of-interest, which each consist of an ID, a title, a description and a URL were included in examples2014.csv. 299 users indicated their preferences by giving a rating on a 5-point score (0, 1, 2, 3, 4) to the description and website of each example point-of-interest. 116 users, indicated preferences to all the 100 example points of interests, these profiles are distributed in profiles2014-100.csv. The other 183 users, only indicated 70% of all the example points of interest, these profiles are distributed in profiles2014-70.csv. There are 50 contexts which each represent a city in the United States which are listed in contexts2014.txt.

Each group is permitted to submit up to 2 runs of suggested point of interests in the 50 contexts. Below we describe in detail our approach for building both of our runs.

2 Gathering Candidate Suggestions

Before analysing users' tastes, we gathered a large selection of possible candidate suggestions. We exploited the resources available on the open web to build a large collection of all possible suggestions. Our collection includes both: a pool of suggestions for every context and a corpus of web pages related to the suggestions.

2.1 Search Engine APIs

Many commercial search engines such as Yelp[1], Google Place[2], Foursquare[3] and TripAdvisor[4] provides friendly APIs for the developers to access their content such as listings, ratings and reviews. We queried these APIs for points-of-interest with contextual information such as the city name, latitude and longitude. This returned point-of-interest results for each context. From the retrieved results, we extracted the name, URL, location, categories, rating, tips and reviews for each point-of-interest. The name and URL are used later to merge point-of-interest data from different API sources. Approximately 120 points-of-interest were retrieved for each context, in total 6000 points-of-interest were extracted.

[1] http://www.yelp.com/developers
[2] https://developers.google.com/places
[3] https://developer.foursquare.com/
[4] https://developer-tripadvisor.com/home/

2.2 Additional Crawling

In order to obtain more information on each candidate suggestion, we also crawled the point-of-interest's website to collect related web pages. The home page of an attraction is expected to contain content that is highly relevant to the attraction. Thus, we parse text content on home web page of website of each candidate suggestion. Moreover, the web pages that home page links to are also expected to be informative. Most of them are within the same website, such as menu, history, introduction, price and contact information of a restaurant. In order to prevent too many pages from being crawled we only crawled pages linked directly from the home page. We also limited our crawl to only the first 50 linked pages (most pages contained less than 50 links), again in order to prevent too many pages from being crawled.

The corpus of web pages along with info supplied by the API (ratings, reviews, categories, descriptions) were used later to extract features for each candidate suggestion.

3 Extracting Features of Candidate Suggestions

The features we extracted for each candidate suggestion were term features. We applied established key word extraction methods to the crawled web pages corresponding to each candidate suggestion, and added some category type terms as special tokens.

3.1 Key Word Extraction

Each point-of-interest is associated with a set of web pages consisting of the home page and linked pages. We combined these pages together to form a large document, our candidate suggestion document (CSD). The home page is assigned a higher weight in this combination, since it is more likely to be relevant. To extract key words from this document, we use pointwise K-L divergence [2].

In order to build a background model, we downloaded all English Wikipedia articles[5]. All page content is tokenized and urls are excluded from the page content. The entire token set is treated as a collection that forms our background model.

We used K-L divergence to rank the tokens by how informative they are. We apply the same method to construct a language model separately from the background Wikipedia articles and each CSD.

The simple language model is the ratio of the number of times a token appears in source s to the total number of times all tokens appear.

$$L_s(t) = \frac{f_t}{\sum_{t \in T} f_t}$$

We computed the divergence between the language model of CSD and the language model of Wikipedia articles. By applying the computation to each single token in CSD, we can rank the tokens based on the divergence score.

$$L_{CSD}(t) * log(\frac{L_{CSD}(t)}{L_{Wiki}(t)})$$

$L_{CSD}(t)$ is the language model of term t in the candidate suggestion document and $L_{Wiki}(t)$ is the language model of term t in all the Wikipedia articles. We take the top-100 tokens from this ranking to form a set of features for each candidate suggestion.

[5] http://dumps.wikimedia.org/enwiki/

3.2 Building a Feature Vector

Our feature vector can be divided into two types of features: plain token features and category features. Top 100 K-L divergence ranked terms fulfill the plain token part of our feature vector. The other group of important features are category data. We gathered category data from those search engine APIs, and here we use it as features. Since our features are all token features, however, we want to differ category features apart from normal token features. A prefix CAT_ is added to category token, which makes it a new and unique token. There can be multiple category terms from the APIs, and each term in this set is included as a category feature into our feature vector.

4 Clustering Candidate Suggestions

Both candidate suggestions and example suggestions are extracted by our feature extraction method. Given the feature vectors of each candidate suggestion and example suggestion (more than six thousand item feature vectors), we applied a form of top-down *Hierarchical K-Means Clustering*. The Sofia-KMeans package[6] is used for our Hierarchical K-Means clustering.

```
Input: Array of Cluster Cluster, Integer Layers
Output: Array of Clusters H_Cluster
if Layers >5 then
    return null;
end
for every cluster c in Cluster do
    if size of c >threshold then
        T = HKM_Cluster(c, Layers+1);
        if T ≠ null then
            insert T into H_Cluster;
        end
    else
        insert c into H_Cluster;
    end
end
return H_Cluster;
```

Algorithm 1: HKM_Cluster($Data, Layers$)

Algorithm 1 describes our Hierarchical K-Means clustering. Here we recursively cluster, if the size of any sub layer cluster is greater than our threshold, we use K-Means clustering to separate it into multiple sub clusters. Also, to prevent infinitely deep clusters we limit the hight of the hierarchical clusters to 5.

Clusters are represented in numbers, such as cluster 3 and cluster 5, cluster names include their parent clusters which are separated with dots (.). If a cluster is divided into several sub clusters, we add a dot after the parent cluster's number, then output the child cluster's number. For example, cluster 7 can be divided into sub clusters: 7.1, 7.2 and 7.3.

5 Matching and Ranking Suggestions

5.1 User Like and Dislike

Each user indicated their preference with a rating from 0 to 4 for both the description and website of each example point-of-interest. In our model, if the user assigned a score of 3 or 4 to the website, it means that the user like this suggestion. Otherwise, if the user assigned a score of 0 or 1 to the corresponding website, it means that the user dislike this suggestion. Based on this metric, since

[6] https://code.google.com/p/sofia-ml/wiki/SofiaKMeans

all example point-of-interests are clustered by our hierarchical K-means clustering method, we can construct a list of liked clusters and a list of disliked clusters.

A liked cluster means that there is at least one liked point of interest contained in the cluster, disliked clustered are calculated in a similar fashion. In the two liked and disliked cluster lists, we maintained both the cluster name and the number of example point of interests in the cluster. Note that since our clustering result is hierarchical, each suggestion's cluster result is the deepest cluster name in the hierarchical structure, such as 12.1.3.5. Thus, we also count its existence in parental clusters, such that a suggestion located in 12.1.3.5 is also counted in 12.1.3, 12.1 and 12. An example of user's liked cluster list is shown in Listing 1.

```
cluster name : number of liked point of interests
...
cluster 7.10 : 2
cluster 16.7.7.4.8.5 :14
...
cluster 9.14: 1
cluster 16.9: 17
cluster 14.14.1: 4
...
```

Listing 1: UserID 700 's liked cluster counting

5.1.1 Low quality profiles

Since the users are crowdsourcing contributors, some of the profiles have poor quality. Some users were not careful when selecting rating scores for the given example point of interests. Therefore, none or little opinion information can be extracted from these profiles. In order to handle these users, we simply looked for the users that assigned the same score for every suggestion. For example, we found that UserID 933 assigned score 2 to every description and website. For this user, we selected the top rated candidate suggestions from each context as our suggestions.

5.2 Selecting Candidates

To suggest a point of interests for each user based on each context, cluster names are grouped by context. For each context, we have a list of clusters and number of suggestions that are assigned to each cluster in this context. All the search engine APIs we used provide rating scores for each suggestion, although with different scales. We normalize the rating scores and compute an average rating for each suggestion. Within a cluster, suggestions are ranked by their average rating. For this task we needed to provide 50 suggestions for each context. When selecting candidate suggestions, we started from the top of ranked clusters, and then choose one candidate from the top cluster. We then proceeded one by one to the bottom of the ranked cluster list. To achieve diversity, we only select suggestions from the clusters that had not been previously selected. If there are not enough suggestions when we reach the last ranked cluster, we start from the top ranked cluster again, until enough suggestions are selected.

5.2.1 Ranking Candidate Clusters

The clusters in each context are ranked for each user. We assign a score for each cluster, which is ratio of the number of likes the user have in this cluster to the number of candidate suggestions in this cluster.

$$\frac{\#like_{user}}{\#candidate\ suggestion}$$

5.2.2 BlackList Candidates

Since we queried the search engine APIs only based on context information and did not filter any results, the candidate suggestions may contain many chains or uninteresting places. When designing our system we made an effort to only make interesting suggestions to users and not suggest, for example, chain stores which are almost the same all over the world. Thus, we first find the candidate suggestions that exist in more than two contexts and add them to our potentially blacklisted set. Then we manually select items from this list, in order to produce our final blacklist. Listing 2 shows part of our blacklist URLs, that we do not wish to suggest to users.

```
...
www.target.com
www.kohls.com
...
www.costco.com
www.academy.com
www.fredmeyer.com
...
```

Listing 2: Candidates in blacklist

5.2.3 Filtering Dislike Candidate Suggestions

In the second run we submit, we add dislike filtering to the candidate selection process. When we rank the clusters in each context for each user, the score assigned to each cluster becomes ratio of the number of liked example suggestions deducted by the number of disliked example suggestions in this cluster, to the number of candidate suggestions in this cluster. The score is limited by users' like and dislike preferences.

$$\frac{\#like_{user} - \#dislike_{user}}{\#candidate\ suggestion}$$

6 Generating Descriptions

In order to generate descriptions for the suggested points-of-interest we took either a short description or short review from the data returned by the APIs. A category was appended to the front of this description in order to quickly give users an idea of what the point-of-interest was when the name and rest of description were unclear.

7 Runs and Results

The two runs we submitted were waterlooA and waterlooB. waterlooA used only positive preferences (see section 5.2.1), waterlooB incorporated both positive and negative preferences (see section 5.2.3). The evaluation metric used for the contextual suggestion task are P@5, MRR, and TBG which are described in last year's overview paper for this track [1].

As can be seen in table 1 using both positive and negative preferences outperformed using only positive preferences. This is similarly reflected in table 2. Additionally from table 2 we can see that for P@5 baselineB got the highest P@5 score given across all runs for approximately 14% ($\frac{41}{299}$) of the context-profile pairs. 82% ($\frac{244}{299}$) of our submitted profile-context pairs got a median or better P@5 score compared to all runs.

8 Conclusion

In this work we had described our system which makes personalized point-of-interest suggestions to users as part of the contextual suggestion track framework. We have seen a method for clustering

Run	P@5	MRR	TBG
waterlooA	0.4167	0.6021	1.7364
waterlooB	0.4308	0.6244	1.8379

Table 1: P@5, MRR, and TBG scores for both of our runs.

Run	Best			Median or better		
	P@5	MRR	TBG	P@5	MRR	TBG
waterlooA	32	147	36	239	233	190
waterlooB	41	152	39	244	239	208

Table 2: Number of profile-context pairs (out of a total of 299) where our runs scored best (or median or better) among all runs.

attractions and determining which clusters of attractions users prefer to have attractions selected from. We have also seen that using both positive and negative preferences from users outperforms using only positive user preferences. Additionally we have seen that we are able to score best among all the runs for about 14% of the profile-context pairs using the techniques described here.

References

[1] Adriel Dean-Hall, Charles L. A. Clarke, Jaap Kamps, Paul Thomas, Nicole Simone, and Ellen Voorhees. Overview of the TREC 2013 contextual suggestion track. In *22nd Text REtrieval Conference*, Gaithersburg, Maryland, 2013.

[2] Takashi Tomokiyo and Matthew Hurst. A language model approach to keyphrase extraction. In *Proceedings of the ACL 2003 Workshop on Multiword Expressions: Analysis, Acquisition and Treatment - Volume 18*, MWE '03, pages 33–40, Stroudsburg, PA, USA, 2003. Association for Computational Linguistics.