Better Contextual Suggestions in ClueWeb12 Using Domain Knowledge Inferred from The Open Web

Thaer Samar[1], Alejandro Bellogín[2], and Arjen P. de Vries[1]

[1] Centrum Wiskunde & Informatica, {samar,arjen}@cwi.nl
[2] Universidad Autónoma de Madrid, alejandro.bellogin@uam.es

Abstract. This paper provides an overview of our participation in the Contextual Suggestion Track. The TREC 2014 Contextual Suggestion Track allowed participants to submit personalized rankings using documents either from the Open Web or from an archived, static Web collection (ClueWeb12) collection. One of the main steps in recommending attractions for a particular user in a given context is the selection of the candidate documents. This task is more challenging when relying on ClueWeb12 collection rather than public tourist APIs for finding suggestions. In this paper, we present our approach for selecting candidate suggestions from the entire ClueWeb12 collection using the tourist domain knowledge available in the Open Web. We show that the generated recommendations to the provided user profiles and contexts improve significantly using this inferred domain knowledge.

1 Introduction and Motivation

The Contextual Suggestion TREC Track investigates search techniques for complex information needs that are highly dependent on context and user interests. Input to the task are a set of profiles (*users*), a set of example suggestions (*attractions*), and a set of contexts (*locations*). Each attraction has a `title`, a `description`, and a `URL`. Each profile corresponds to a single user, and indicates the user's preference with respect to each attraction. Two ratings are used: one for the attraction's description and another one for its website. Finally, each context corresponds to a particular geographical location (a city and its corresponding state in the United States). With this information, the task is to provide a personalized ranked list of up to 50 suggestions for every (user, context) pair. Each suggestion should be appropriate to both the user's profile and the context. The description and title of the suggestion may be tailored to reflect the preferences of that user.

In our second year participating in the Contextual Suggestion track, our main goal has been to analyze the impact of applying the same retrieval model on two collections designed differently. We submitted two runs: one based on a collection which has high geographical precision (GeoFiltered) and another based on a collection created with high recall after projecting Open Web tourist domain knowledge on ClueWeb12 collection (TouristFiltered).

In the rest of this paper, we present how we generated these sub-collections, their statistics (Section 2), and, results obtained in each case (Section 3). Finally, we conclude the paper with future work lines and general conclusions (Section 4).

2 Methodology

In this section, we describe our approach for generating a personalized rank list of suggestions for each (user, context) topic. First, we present our approach for selecting candidate documents from ClueWeb12 collection. Then, we describe how we generated user's profiles based on the descriptions of the attractions rated by her. For each user, we generate a positive and a negative profile. Then we represent the set of candidate documents and the profiles in the $|V|$-dimensional vector space ($|V|$ is the size of the vocabulary), where each element in the vector is a pair of term id and the frequency of that term in the document [3]. Finally, we rank the suggestions based on their similarity with user's profiles.

2.1 Generating the Set of Candidate Documents

In this section, we describe our approach for creating two sub-collections from ClueWeb12 collection. The first sub-collection named **GeoFiltered** was created by extracting all documents from ClueWeb12 that pass a geographical filter. The second sub-collection named **TouristFiltered** was created by exploiting the tourist domain knowledge inferred from the Open Web.

Geographically Filtered Sub-collection For this sub-collection, we extracted all documents from ClueWeb12 that pass a geographical filter `geo_filter`. This filter selects documents that mention the contexts given by the organisers in the format `(City, ST)`, ignoring those mentioning the city with different states or match multiple contexts. After this process, our subcollection contained $8,883,068$ documents that passed the filter where each document mention only one context. Thereby, we want to make sure that the filtered documents mention the specific target context (hence, being geographically relevant documents). Despite this, we might still miss other relevant documents due to misspellings or because they mention more than one city at the same time. We name this subcollection `GeoFiltered`.

Tourist Filtered Sub-collection This sub-collection was created from ClueWeb12 collection using three different filters using the tourist domain knowledge available on the Open Web as described below:

1. **Tourist sites:** We manually selected a list of sites that are well-known of providing tourist information. More precisely, the list consists of `(yelp, tripadvisor, wikitravel, zagat, xpedia, orbitz, and travel.yahoo)`.Then we extracted any document from ClueWeb12 whose host is one of the tourist sites. This part of the TouristFiltered sub-collection is called **TouristListFiltered**.
2. **Tourist outlinks**: To complement the collection, we extracted the outlinks from all documents collected in step 1 and then we extracted those outlinks from the ClueWeb12 collection. We name this part of the TouristFiltered sub-collection **TouristOutlinksFiltered**.

3. **Google and Foursquare APIs:** This part was created in two steps. We first queried Foursquare API [1] to get venues for the given contexts, the API returns 50 urls per query. If the returned venue does not have a URL, then we use the Google Search API to get the URL. The query is the venue name and the context. As a second step, we extracted all the URLs found by Foursquare and Google API from ClueWeb12. We only found `234 out of 2316` documents by exact URL matching after normalizing the URL by removing the `www, http://, or https://`. The number of documents is very low and actually for some contexts no document was found. Therefore, to alleviate this coverage problem, we extracted the host information from each of the URLs returned by Foursquare and Google APIs, and then we used any document from ClueWeb12 matching these hosts. These documents form what we called the **AttractionFiltered** sub-collection.

Table 1 shows the number of documents extracted for each sub-collection, whereas Table 2 gives more details about the TouristFiltered collection parts. It shows the total number found by each TouristFiltered part filters. The unique column represents the total number of new documents found by the filter that are not already found by the previous filter starting with the TouristListFiltered filter, then TouristOutlinksFiltered, and finally the AttractionFiltered filter. We observe that most of documents in the TouristFiltered sub-collection came from the TouristListFiltered and the AttractionFiltered. There is a big overlap between TouristListFiltered and TouristOutlinksFiltered (around 52%). This is because TouristOutlinksFiltered consists of outlinks of TouristListFiltered documents, which contains both external links (link to a page from different host), and internal links (link to a page from the same host) which means that these documents are already in the TouristListFiltered part.

Table 1. Number of documents per collection.

sub-collection	Number of documents
GeoFiltered	8,883,068
TouristFiltered	324,374

Table 2. Number of documents for each part of the TouristFiltered sub-collection.

Part	Number of documents	Unique (not already in)
TouristListFiltered	175,260	175,260
TouristOutlinksFiltered	97,678	46,801
AttractionFiltered	102,604	102,313
Total	375,542	324,374

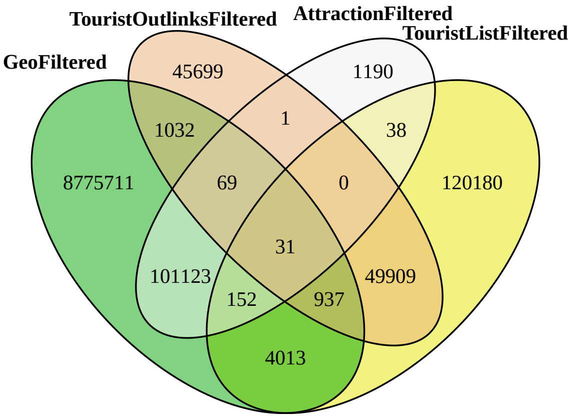

Fig. 1. Intersection between GeoFiltered collection and TouristFiltered collection parts.

Figure 1 shows the overlap between TouristFiltered parts and the GeoFiltered subcollection.

2.2 Generating User Profiles

We generated a textual user's profile using the descriptions of the attractions rated by the user. The ratings of the descriptions where used to split the user's profiles into positive and negative profiles. The ratings are on a 5-point scale, each rating represents a user's level of interest in visiting the corresponding attraction, the levels ranging from "0" for strongly uninterested to "4" for strongly interested. In this context, we consider the "2.5" as threshold between negative and positive ratings. More precisely, the positive profile consists of all descriptions of the attractions that the user liked, whereas the negative profile is the concatenation of descriptions of the attractions that the user disliked.

2.3 Representation of Documents and User Profiles

To represent the candidate documents and user profiles in the Vector Space Model (VSM), we first filtered out the HTML tags from the content of the documents. Then we applied standard IR parsing techniques including stemming and stop-words removal from the documents and the user profiles. Once the documents and profiles have been parsed, we generate a dictionary containing a mapping between terms and their integer ids. Finally, we use this dictionary to transform the documents into vectors of weighted terms, where the weight of each dimension (term) is the standard term frequency *tf*.

2.4 Personalizing Rankings

To generate the final ranking (given a pair of context and user information), we compute the similarity in the vector space representation between the document and both the positive and the negative user profiles. We used the cosine function to compute similarities between candidate documents and both the positive and negative profiles as follows:

$$sim(u^{+}, d) = \cos(u^{+}, d) = \frac{\sum_i {u_i}^{+} \cdot d_i}{\sqrt{\|u^{+}\|_2}\sqrt{\|d\|_2}} \tag{1}$$

$$sim(u^{-}, d) = \cos(u^{-}, d) = \frac{\sum_i {u_i}^{-} \cdot d_i}{\sqrt{\|u^{-}\|_2}\sqrt{\|d\|_2}} \tag{2}$$

The final score is based on these two similarity scores using the following equation:

$$score = a \cdot sim(u^{+}, d) + b \cdot sim(u^{-}, d) \tag{3}$$

where a=2 and b=-1. The final score in Eq(3) was used to rank the suggestions per (user, context) pairs.

2.5 Generating Descriptions and Titles

For each document suggested to a user in a context, we generate a description and title, which would be tailored to the particular user and context if possible. We decided to only provide personalized descriptions, since we consider the title as a global property of the document, inherent to its content and, thus, should not be different for each user. In this situation, we generate the titles by extracting the title or heading tags from the HTML content of the document. On the other hand, we observe the task of generating descriptions similar to snippet generation where the query is the combination of context and user preferences [2]. Because of that, we aim at obtaining the most relevant sentences for the user within the document in a particular context. To do this, we first split the document into sentences by using the Java BreakIterator class[3] which can detect sentence boundaries in a text. We then followed similar steps to those of the document ranking but at a sentence level, i.e., filter out those sentences not mentioning the context; and, we extracted the text of the $description$ tag from the HTML content. Then we rank sentences according to their similarity with the user profile. Finally, we assumed that larger descriptions were preferred, and hence, we concatenated sentences – in decreasing order of similarity – until the maximum number of bytes (512) was reached, controlling to not combine two very similar sentences to decrease the redundancy.

[3] `http://docs.oracle.com/javase/6/docs/api/java/text/BreakIterator.html`

3 Results and Analysis

In this section we present the analysis of the performance of our runs compared to all runs based on ClueWeb12 collection. We present a detailed comparison between our two runs to show the effect of applying the domain knowledge on extracting touristic related documents from ClueWeb12. Table 3 shows the performance of our runs (both using the same scoring function presented in Eq 3 but with a different set of candidate documents (either GeoFiltered or TouristFiltered) compared to the best and median scores of all runs based on the ClueWeb12 dataset. From the Table we see that the TouristFiltered run outperforms the GeoFiltered run in the three metrics.

Table 3. Effect of the GeoFiltered and TouristFiltered collections on the performance of our retrieval model. The P@5, MRR and TBG of our runs and the median and best scores of the same metrics for all runs based on ClueWeb12.

	P@5	MRR	TBG
GeoFiltered	0.0468	0.0767	0.1256
TouristFiltered	0.1438	0.2307	0.6013
Median	0.0542	0.0886	0.1382
Best	0.2328	0.4232	0.9615

Additionally, to get a more fine grained comparison about the effect of the two subcollections on the performance, in Table 4 we present the percentage of topics for which the runs associated to the TouristFiltered and GeoFiltered subcollections give the best and the worst results (we consider only topics having a best score different to its worst score). We observe that, for instance, for the TBG metric, the TouristFiltered subcollection gives the best result for 28% of topics, whereas the GeoFiltered subcollection gives the best result for only 9% of topics; at the same time, the TouristFiltered subcollection gives the worst result for 23% of the topics, whereas the GeoFiltered subcollection gives the worst result for 49% of the topics. Hence, it is clear that these two subcollections produced very different results, where the TouristFiltered subcollection outperforms the GeoFiltered one.

Table 4. Percentage of best and worst topics per run for P@5, MRR and TBG metrics. Bold denotes best value per column.

	P@5		MRR		TBG	
	best	worst	best	worst	best	worst
GeoFiltered	9.03	41.14	8.70	41.14	9.03	49.16
TouristFiltered	**28.43**	**20.07**	**25.42**	**20.07**	**28.43**	**23.41**

All the analyses presented so far confirm that the TouristFiltered run is significantly better than the GeoFiltered run in general. The three metrics P@5, MRR and TBG consider three dimensions of relevance, the geographical `(geo)` and profile relevance (both in terms of document `(doc)` and description `(desc)` judgments). In Table 5 we show how each run performed in the three relevance dimensions. The performance of the GeoFiltered run is on par with TouristFiltered when considering the document and the description relevance, this means that both are similar in terms of their appropriateness to the users. However, we observe a significant difference between TouristFiltered and GeoFiltered when considering the geographical aspect only. The TouristFiltered sub-collection is more geographically appropriate, implying that applying our domain knowledge on the sub-collection creation improves the performance with respect to the geographical dimension of relevance.

Table 5. Contribution of description (desc), document (doc), and geographical (geo) relevance to P@5 and MRR metrics for the GeoFiltered and TouristFiltered runs. We denote with (all) when desc, doc, and geo relevance are considered. Bold highlights the best result for each run and metric.

Metrics	GeoFiltered	TouristFiltered
P@5_all	0.0468	0.1438
P@5_desc-doc	0.2281	0.2348
P@5_desc	**0.3064**	0.2910
P@5_doc	0.2836	0.3124
P@5_geo	0.1605	**0.4843**
MRR_all	0.0767	0.2307
MRR_desc-doc	0.2987	0.3647
MRR_desc	**0.3942**	0.4408
MRR_doc	0.3701	0.4736
MRR_geo	0.2231	**0.6527**

Finally, to provide a deeper insight into the question why the domain knowledge-based sub-collection improves so much over the other sub-collection on the different relevance dimensions, we present in Table 6 the contribution to the relevance dimensions of each of the sub-collections that take part to build the TouristFiltered sub-collection (recall that it consists of three parts: TouristListFiltered, TouristOutlinks-Filtered, and AttractionFiltered). Note that the TouristFiltered sub-collection corresponds to the third column, where the three parts are combined.

To obtain these results, we modified the submission file of the run based on the TouristFiltered sub-collection. First, we compute the performance with keeping only suggestion in the run file that are from TouristListFiltered. Then we added those coming from the TouristOutlinksFiltered part, and finally those from AttractionFiltered. As we observe in the table, there is a major improvement after adding the AttractionFiltered part. As a final comparison, we also include the performance of the AttractionFiltered part alone; we observe that its suggestions are almost comparable to the whole sub-collection but not as good as those, in particular because of the description relevance dimension.

Table 6. Effect of TouristFiltered sub-collection parts on P@5 and MRR metrics. We denote TouristListFiltered, TouristOutlinksFiltered, and AttractionFiltered parts as TLF, TOF, and AF respectively. In bold, the best result per column and metric; the best overall result is underlined.

Metrics	TLF	TLF + TOF	TLF + TOF + AF	AF
P@5_all	0.0314	0.0441	0.1438	0.1084
P@5_desc-doc	0.0609	0.0856	0.2348	0.1599
P@5_desc	0.0736	0.1023	0.2910	0.2007
P@5_doc	0.0809	0.1110	0.3124	0.2161
P@5_geo	**0.1612**	**0.2181**	**0.4843**	**0.4468**
MRR_all	0.1101	0.1453	0.2307	0.1843
MRR_desc-doc	0.1782	0.2339	0.3647	0.2823
MRR_desc	0.2101	0.2671	0.4408	0.3644
MRR_doc	0.2259	0.2947	0.4736	0.3759
MRR_geo	**0.4132**	**0.4841**	**0.6527**	**0.6047**

4 Conclusions and Future Work

In our submission this year we focused on extracting a set of tourist documents from the ClueWeb12 collection. We discussed our approach of selecting candidate documents from the ClueWeb12 collection, using tourist domain knowledge inferred from the Open Web in order to get better suggestions from ClueWeb12 collection. We measured the impact of using the domain knowledge by applying the same retrieval model on two sub-collections created differently; one using the domain knowledge, while the other does not have this knowledge. Based on the analysis of the experimental results, we have observed that the sub-collection that uses the domain knowledge significantly improves the performance of the retrieval model.

We applied domain knowledge for creating a subcollection which consists of parts, were each part was defined as a Boolean filter. For the future work we can extend our filter model towards weighted variants based on the content and the domain knowledge. Also in the future we want to study if there are properties of the sub-collection – or the documents that shape it (length, vocabulary distribution, etc.) – that allow to infer whether a subcollection will be useful for a particular task, even before running the retrieval method.

Acknowledgments

This research was supported by the Netherlands Organization for Scientific Research (NWO project #640.005.001)

5 References

[1] Foursquare api. https://developer.foursquare.com/docs/venues/search.

[2] H. Luhn. The automatic creation of literature abstracts. *IBM Journal of Research and Development*, pages 159–165, 1958.

[3] G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, 1975.