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, the ClueWeb12 dataset. In this paper, we focus on filtering the entire ClueWeb12 collection to exploit domain knowledge from touristic sites 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 includes a title, a description, and an associated 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 title and 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, up to 50 ranked suggestions should be generated by each participant for every context and profile 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 geographical context knowledge on ClueWeb12 collection (TouristFiltered).

In the rest of this paper, we present how we generated these subcollections, their statistics (Section 2), and, results obtained in each case (Section 3), which allowed us to be among the top performing teams. Finally, we conclude the paper with future work lines and general conclusions.

2 Methodology

In this section, we describe our approach for generating a list of suggestions for each user profile and geographical context. The first step is the selection of candidate documents from ClueWeb12 collection that are geographically appropriate. Then, we generated user profiles based on the descriptions of the attractions rated by them. We took the ratings of the descriptions into consideration. Therefore, for each user we generated positive and negative profiles. Then we represented the set of candidate documents 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]. After that, we computed the similarity between the candidate documents and both the positive and the negative profiles. Finally, we ranked the suggestions based on the document-user similarity score.

2.1 Generating the Set of Candidate Documents

In this section, we describe how we designed the collection of documents for each run. First, a collection named **GeoFiltered** was created by extracting all documents from ClueWeb12 that pass a geographical filter. This geographical filter is looking for the documents that mention the contexts given by the organizers in the format `(City, ST)`, ignoring documents that mention the city with different states or match multiple contexts. Second, we created another subcollection from ClueWeb12 that we named **TouristFiltered**. The TouristFiltered subcollection has been created by applying domain knowledge on ClueWeb12 in three steps:

1. **Tourist sites:** We manually selected a list of sites that we think are tourist sites. 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 subcollection 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 subcollection **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** subcollection.

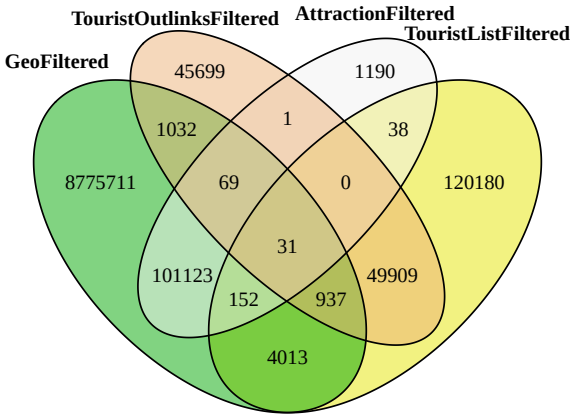

Fig. 1. Intersection between GeoFiltered collection and TouristFiltered collection parts.

Table 1. Number of documents per collection.

Dataset	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

Table 1 shows the number of documents extracted for each collection, whereas Table 2 gives more details about the TouristFiltered collection parts. We observe that TouristListFiltered and AttractionFiltered are the steps where unique documents are found. It is interesting to note that as a result of our process, we are able to recover from ClueWeb12 context-relevant documents that not necessarily mention the context, in particular those coming from AttractionFiltered. We observe this aspect in detail in Figure 1, where the overlap between the GeoFiltered collection and the three parts of the TouristFiltered collection is shown. In the Figure we can see how TouristListFiltered and TouristOutlinksFiltered provide a large number of new documents to GeoFiltered, this is mainly because these steps do not take the target context into account.

2.2 Generating User Profiles

We generated each user's profile according to the user's preference for the given attractions and the descriptions of those attractions. For each user we generated a positive and negative profile based on the descriptions ratings. 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 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=1 and b=-2. 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, besides this we extracted 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 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. In Figures 2, 3, and 4 we present a comparison of the two runs and the median for the P@5, MRR, and TBG metrics, respectively. Figure 2 shows the number of topics where a relevant document was returned in the top 5 positions, a zero means there was no relevant document in the top 5. Figure 3 shows the number of topics containing one relevant document observed at position x in the top 5. Figure 4 shows the TBG (time-biased gain) performance of the two runs. From these figures we conclude that the TouristFiltered run consistently outperforms the GeoFiltered run, and usually obtains better performance than the median.

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

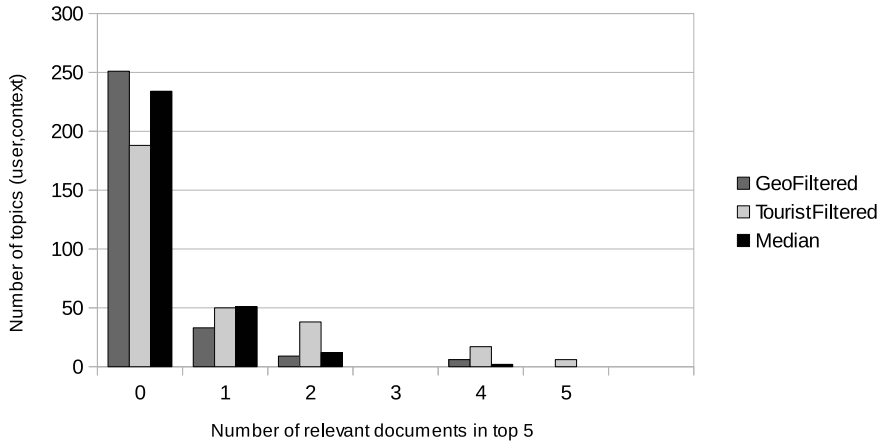

Fig. 2. Number of topics per top_k.

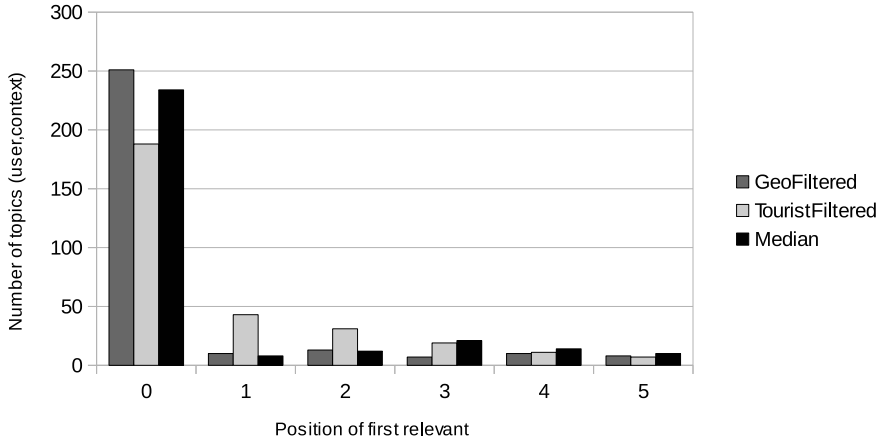

Fig. 3. Number of relevant documents for each topic at each position in the top 5.

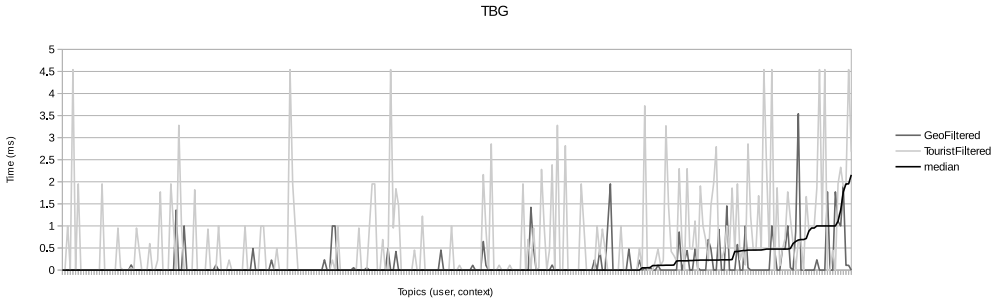

Fig. 4. TBG time for all topics, i.e. (user, context) pairs.

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

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**

Table 5. Comparison between the two runs by showing the percentage of topics where the TouristFiltered run gives better, equal, or worse performance compared to the GeoFiltered run.

	GeoFiltered			
	Better	Equal	Worse	Metric
TouristFiltered	33.11	58.53	8.36	P@5
	32.44	58.53	9.03	MRR
	41.47	47.49	11.04	TBG

To get more insights into the factors determining the performance of our runs, we compute the percentage of topics for which our runs gave the best and worst results; we consider only topics whose best score is not equal to the worst score. Table 4 shows the percentage of best and worst topics for all metrics. From the percentage of best and worst topics per run, we see that the TouristFiltered run obtains more frequently better performance than the GeoFiltered run. Moreover, in Table 5 we present a one-to-one comparison between the TouristFiltered and GeoFiltered runs based on number of topics where TouristFiltered run is better than, equal to and worse than GeoFiltered run for the three metrics.

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 6 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

Table 6. 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.

Metric	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**

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 subcollection is more geographically appropriate, implying that applying our domain knowledge on the subcollection creation improves the performance with respect to the geographical dimension of relevance.

Finally, to provide a deeper insight on why the domain knowledge-based subcollection improves so much over the other subcollection on the different relevance dimensions, we present in Table 7 the contribution to the relevance dimensions of each of the subcollections that take part to build the TouristFiltered subcollection (recall that it consists of three parts: TouristListFiltered, TouristOutlinksFiltered, and AttractionFiltered). Note that the TouristFiltered subcollection corresponds to the third column, where the three parts are combined.

To obtain these results, we started by keeping in the submission file only the suggestions coming from the TouristListFiltered part, then we added those coming from the TouristOutlinksFiltered part, and finally those from AttractionFiltered part. 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 subcollection but not as good as those, in particular because of the description relevance dimension.

Table 7. 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.

Metric	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 creating a set of touristic documents from the ClueWeb12 collection. We introduced the domain knowledge of Open Web for extracting candidate tourist documents from ClueWeb12. The analysis of the performance of our GeoFiltered and TouristFiltered runs showed that applying domain knowledge may lead to performance improvements of the retrieval model. In the future, we will test other ways for extracting touristic documents from the ClueWeb12 collection, including the use of different sources of knowledge from the Open Web. We aim also to experiment with other scoring functions, in order to better capture the user's preferences.

Acknowledgements

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.