University of Delaware at TREC 2014

Ashraf Bah, Karankumar Sabhnani, Mustafa Zengin, and Ben Carterette
Department of Computer and Information Sciences, University of Delaware, Newark DE USA
[ashraf | karans | zengin | carteret]@udel.edu

Abstract

This paper describes the work of the Information Retrieval Lab at the University of Delaware (team name "udel") on TREC 2014 tracks. We participated in five different tracks: Contextual Suggestion, Federated Web, Microblog, Session, and Web.

1 Introduction

The Information Retrieval Lab at the University of Delaware participated in five different tracks at TREC 2014. This paper describes our work in those tracks. Sections are organized as follows (with primary author(s)/contact person):

Section 2 Contextual Suggestion track, by Karankumar Sabhnani.

Section 3 Federated Web track, by Mustafa Zengin.

Section 4 Microblog track, by Karankumar Sabhnani.

Section 5 Web & Session tracks, by Ashraf Bah and Ben Carterette.

2 Contextual Suggestion Track

Our system tackles the contextual suggestion task in multiple phases. The first phase—the learning phase—is where we analyze user profiles and generate bags of keywords depicting the user interests. The second phase—retrieval—is where given a context, we query the Google Places API with each bag and retrieve lists containing places in that context which fit in the specified genre. We then aggregate all sets of results for a single user's profile into one set of 50 ranked results, which is the third and the final phase.

This approach, which we also used for last year's track [1], worked very well, achieving highest performance for most of last year's user-context pairs and performance above the median for most. However there were a few pairs for which it generated poor suggestions. We did some analysis to investigate the reason behind this behavior and came up with some techniques that could boost our performance. This report summarizes, 1) the new components in our approach, 2) submitted runs, and 3) performance evaluation.

2.1 Keyword ranking and description generation

Our analysis indicates that the learning and retrieval phases work well. To improve the performance, we should optimize the order in which the top 50 places are picked. This tells us that the ordering of keywords within the bag of each profile can improve the results significantly.

Another thing that we didn't address in last year's runs was the generation of description for every suggestion. Our current approach incorporates both of the above-mentioned ideas.

2.1.1 Keyword ranking

One way of ordering keywords within the bag is by assigning weights to each keyword while learning. The weight could be as simple as the number of times the user has liked that category. Ordering the keywords in descending order, i.e. the keyword with the highest weight first, will make sure that places from categories that a user prefers more will occur at higher ranks.

This approach didn't actually work as the data was biased towards particular categories. Out of the 50 training examples from 2013, there are around 10 museums, 5-7 landmarks, 4 theaters and a couple of breweries; because of this, the top 5 keywords for all the profiles turned out to be museums, landmark, theater, breweries/spas/tours/markets. This year's data has more examples and two different seed cities, so it is a little less biased, but still won't help much as only top 5 places are evaluated and the top 5 keywords for most of the profiles turn out to be the same.

As it was not feasible to rank all the keywords, we prioritized some of the common keywords such as restaurants, museum/landmark, desserts, etc. and moved them up in the list if they already exist in the bag. This will make sure that we don't present the users with bad categories at top ranks.

2.1.2 Description generation

Descriptions play an important role as the user generally reads and rates them first before visiting the place website and are equally important as the website ratings. We generate descriptions for each place suggestion using categories, aspects, and user reviews (all returned by Google Places) of that place. Aspects include serivce, decor, food, etc.; we try and include aspects for which the place is recognized. Along with its best aspects, we also include one review and its overall Google rating, if available. All this information is retrieved using Google Places API and then clubbed to create sentences in two different ways. Here are two examples of actual descriptions that were generated and included in our runs:

> Sweet Beginnings Bakery, a bakery in Buffalo,NY is highly appreciated for its quality, appeal, and service with service being the best aspect. Here's one customer review: Everytime I buy something there it is the best quality I have ever seen and I have been to a lot of bakeries. The best in the Buffalo area by far.;.

> Bunk Morrison is one of the best sandwich places in Portland,OR. It is know for its food, service, and decor. In fact, its decor makes it work visiting. The overall google rating of this place is 4.3. Here's one customer review: Yes... word around Portland is true: Bunk Sandwiches are amazing!!! I have dreams at night, and cravings in the day, about the chicken salad sandwich. Also, it is entirely true that one breakfast sandwich can be THAT much better than the others... grand central bread, top qualit

This process is fully automated and one of two ways to generate and club sentences is randomly selected.

2.2 Submitted runs

Our main idea is to maintain diversity among all the genres that suit the user's taste. We have submitted two runs based on this approach. For both the runs, we start by analyzing user profiles

and generating bags of keywords depicting the user interests. Then, given a context, we query Google Places API with each bag, retrieving lists containing places in that context which fit in the specified genre. The API results are used for generating descriptions for each suggestion and top 50 suggestions for every profile-context pair are then used for creating our final run. The only difference in both the runs is that we prioritized keywords in each bag for the first run.

2.3 Evaluation

Our first run (**run_FDWD**) performs much better on average compared to all our runs submitted so far for this track at TREC. It has an average of 0.4201 for P@5, MRR of 0.5649 and TBG of 1.7589; it is as good as the best performance for some profile-context pairs and performs better than the median for the rest. These new components improved the performances by approximately 15%.

The second run (**run_DwD**) has an average of 0.3097 for P@5, MRR of 0.3760 and TBG of 0.9532. It does not perform well and this clearly indicates that the ordering of keywords plays an important role.

3 Federated Web Track

The Federated Web track this year included three tasks: vertical selection, resource selection, and results merging. We submitted runs for vertical selection (Section 3.2) and resource selection (Section 3.1).

3.1 Resource Selection

We submitted two runs (**udelftrssn, udelftrsbs**) to the resource selection task. For both of the runs we employed the same method on snippets and sampled pages, in order to rank resources according to their relevance to a given query. We created two central indexes; one for snippets and one for sampled pages. For each query, we ranked the documents according to their query-likelihood score and selected the top 100 documents for further processing. Then the resources were ranked according to the number of documents they contributed to the selected documents. Table 1 shows the performances of the same algorithm on snippets (udelftrssn) and on sampled pages (udelftrsbs).

RunID	nDCG@10	nDCG@20	nP@1	nP@5
udelftrssn	0.174	0.216	0.147	0.149
udelftrsbs	0.272	0.355	0.166	0.255

Table 1: Resource Selection Runs

Based on these results, ranking based on the full sampled pages is unequivocably better than just using snippets.

3.2 Vertical Selection

We submitted two runs to vertical selection task and in both runs we used the **udelftrsbs** resource selection run. Our baseline run, **udelftvql** ranked verticals according to the number of resources ranked for **udelftrsbs**.

For the **udelftvqlR** run, we applied certain rules to modify and re-rank **udelftvql**. We assigned two scores to verticals. The first score, s_1, is the score that a vertical gets from the rules. The second score, s_2, is the negative rank of the vertical in **udelftvql**. The rules are as follows:

- If a query starts with an interrogative word (e.g. *who, what, where, why, when*), Q&A and General verticals should be added to the verticals list. Their s_1 scores were set to 100 and 80 respectively.
- If a query does not contain any interrogative word, Q&A vertical should be removed from the verticals list.
- Vertical words ("video" for video vertical, "shopping" for shopping vertical, and so forth) should be searched for in a query. Vertical names that contain query terms should be added to the verticals list. Their s_1 scores should be set to 90.

After applying the rules, we sort the verticals according to s_1 and broke the ties with s_2. If no rules apply, then we simply default to the original ranking. We then pick the top 5 verticals for each query. Table 2 below shows the performance of udelftrsbs and udelftvqlR.

RunID	P	R	F1
udelftvql	0.167	0.852	0.257
udelftvqlR	0.236	0.680	0.328

Table 2: Vertical Selection Runs

Here we see that using our rules improves precision by a substantial amount (over 40%), though hurts recall (by 20%). The result is a strong improvement in F1 of 27%.

Rules applied to 40 of the 50 queries (10 did not match any of the conditions). We analyzed how selecting only the top 5 verticals from udelftrsbs and using our rules affects the performance on these 40 queries. Table 3 below shows the performance of udelftrsbs, udelftvqlR and a run that only consists of top 5 verticals from udelftrsbs on 40 queries.

RunID	P	R	F1
udelftvql@40Q	0.156	0.857	0.245
udelftvqlTOP5@40Q	0.218	0.602	0.297
udelftvqlR@40Q	0.233	0.642	0.317

Table 3: Vertical Selection Runs on 40 Query

It can be seen that picking top 5 verticals for each query (the udelftvqlTOP5@40Q run) improves precision by 40%, F1 by 21%, and hurts recall by 30%. Picking top 5 vertical and applying rules (udelftvqlR@40Q) improves the performance further. Improvements in precision and F1 reach to 49% and 30% respectively. Both improvements are significant by a paired two-sided t-test.

4 Microblog Track

Tweet Timeline Generation (TTG) is a new task for Microblog track at TREC 2014. The TTG task attracts us as it supplements the standard challenges of ad hoc retrieval with issues from topic detection and tracking (TDT), and multi-document summarization. Our system efficiently detects

and eliminates redundant tweets and also maintains a good balance between the cluster precision and cluster recall. For every topic, we start by retrieving relevant tweets from the official search API. Then, a subset of these tweets is used for generating semantic clusters using the Quality Threshold (QT) algorithm. Once we have the clusters, we pick exactly one tweet from every cluster that represents it the best and use those to present the user with a list of chronologically ordered tweets that summarize their respective topic/event.

4.1 Extracting relevant tweets and selecting subsets for clustering

We use the official search API to retrieve top 2000 relevant tweets for every topic. Only a subset of these tweets is used for clustering. Scores and ranks provided by the search API are used to select tweets up to a certain score threshold subject to a fixed number of minimum (75) and maximum (300) tweets. Our goal with this threshold and cap on number of tweets considered for clustering is to help in ensuring a good precision, whereas having a constraint on the minimum number of tweets will hopefully prevent the recall from dropping. This process helps in maintaining a decent balance between the cluster precision and cluster recall.

4.2 Creating semantic clusters

The quality threshold algorithm with complete linkage distance is used for creating clusters [2]. Every subset is first sorted on the basis of the tweet time. The first cluster is built with the first tweet in the subset. As long as other tweets are close enough to be within the specified diameter, they are added to the cluster. Once all tweets have been read, the tweets that have been added to the cluster are set aside and the algorithm is repeated recursively on the rest of the tweet collection. The distance between a tweet and the cluster is computed using the complete linkage distance, i.e. the distance from the tweet and the furthest tweet in the collection.

4.3 Selecting cluster representatives and generating summary

The next step is to select exactly one tweet from each cluster that represents it the best and then use them to present the user with a list of chronologically ordered tweets that summarize their respective topic/event. One way is to pick the earliest single tweet in each cluster. Summaries generated with these tweets will truly represent the chronological order of events. Another way is to pick the single highest-scoring tweet from each cluster and then sort them on the basis of their tweet time. This will ensure that the summaries are generated with the most likely relevant tweets. For simplicity, in this year's TTG task, all tweets from a cluster are considered equivalent and thus this step won't really affect the performance of our system.

4.4 Submitted runs

We have submitted 4 runs for the TTG task and 1 run for the ad hoc task. For the ad hoc task, we have filtered retweets and non-english tweets.

Our first two runs for the TTG task use our ad hoc run and the last two use the baseline run as the feed for clustering. The tweet collection for every topic is a subset of the tweets retrieved for the ad hoc or the Baseline run respectively. We set the minimum and maximum size of the tweet collections to be 75 and 300 respectively. The score threshold that we have used for all the TTG runs is determined by taking the score of the tweet at rank 15, then subtracting 3.5 from that (these values were picked manually with no training). The cluster representatives for the first and

Runtag	Rad	Max	Min	Rt	UR	WR	P
TTG1.3	0.24	300	75	3.5	0.1873	0.3645	0.2793
TTG2.3	0.24	300	75	3.5	0.1845	0.3548	0.2752
TTG3.3	0.24	300	75	3.5	0.1832	0.3575	0.2782
TTG4.3	0.24	300	75	3.5	0.1900	0.3697	0.2779
TTG1.6	0.24	600	75	3.5	0.2223	0.3971	0.2319
TTG6.6	0.24	600	75	2.0	0.2065	0.3833	0.2606
TTG9.6	0.20	600	75	2.0	**0.2588**	**0.4454**	0.2550
TTG9.3	0.20	300	75	2.0	0.2349	0.4174	0.2780
TTG10.3	0.22	300	75	2.0	0.2172	0.3951	**0.2865**

Table 4: Unweighted Recall (UR), Weighted Recall (WR), and Precision (P) values for different values of Cluster Radius (Rad), Maximum and Minimum number of tweets considered for clustering (Max & Min), and the Relevancy threshold (Rt). The first four lines are our official submissions; the last 5 are additional experiments.

third run are the earliest tweets from every cluster, and the cluster representatives for the second and the fourth run are the highest-scoring ones.

4.5 Evaluation

Our ad hoc run (**udelRunAH**) had an average precision of 0.1841, precision at 30 of 0.5103, and R-precision of 0.2485. These are not particularly good (or bad).

The first four runs in Table 4 are the TTG runs that we submitted to TREC. Despite the fact that they use different inputs (our ad hoc run for TTG1.3 and TTG2.3 vs the official baseline for TTG3.3 and TTG4.3) and different approaches to drawing from clusters, they all have very similar (around average) performance. Their feeds are actually almost identical in nature and to keep the evaluations simple, the effect of cluster representatives is ignored.

One reason for low recall and precision values could be the feed itself. Both the baseline ad hoc run and our ad hoc run have below average precision and recall values, which indicates that the feed itself has many non-relevant tweets, which in turn results in unwanted clusters and low precision. Given a good feed, our system may yield clusters with much better precision and recall.

Another good aspect of our approach is that all our runs very well balance the ratio of cluster precision to cluster recall. To confirm this fact, we did some additional experiments with parameter values.

On increasing the number of maximum tweets considered for clustering to 600 (TTG1.6), recall increased but precision dropped. However, increasing the score bound by 1.5 (subtracting 2.0 rather than 3.5 from the 15th-ranked tweet's score) (TTG6.6) compensates for this loss of precision. Another parameter that can affect the recall is the similarity radius of the cluster. Reducing it to 0.2 (TTG9.6) boosts the recall to 0.4454.

Overall, the best performance seems to be achieved with a cluster radius of 0.2 and a score bound of 2 less than the 15th-ranked tweet, with the constraint that at most 600 tweets be taken for clustering. This indicates that these parameters indeed help in maintaining the ratio of recall to precision and given a good feed, our approach can create clusters with better precision and recall.

5 Session and Web Tracks

Our runs submitted to the Session and Web tracks used the same indexes and similar methods. The indexes are the same one we used for last year's submissions [1].

5.1 Method 1: Field & Phrase Features

For this method, we use the indri query language to compute language model scores for title, inlink, and url fields for independent query terms as well as an unordered phrase containing all query terms, and combine them with a basic full-document language modeling query. The weights assigned to each feature are hand-tuned.

An example indri query is as follows:

```
#combine( #weight(1   #combine(identifying spider bites)
                  1   #weight(1 #combine(identifying.title spider.title bites.title)
                              1 #uw(identifying spider bites).title)
                  50  #weight(1 #combine(identifying.inlink spider.inlink bites.inlink)
                              1 #uw(identifying spider bites).inlink)
                  0.5 #weight(1 #combine(identifying.url spider.url bites.url)
                              1 #uw(identifying spider bites).url)))
```

These queries were submitted to two different ClueWeb12 indexes to generate two different runs. The first, **udel_itu**, used an index that represents documents by only their title, anchor text from inlinks, and url. The second, **udel_itub**, used an index that contains full document text in addition to those three fields.

We submitted **udel_itu** runs to both the Session and Web tracks. We submitted a **udel_itub** run to the Web track but not the Session track due to time constraints.

5.2 Method 2: CombCAT Fusion

CombCAT is a variation of CombMNZ wherein we not only take the overall similarity values into account, but more importantly we also account for the frequency of a document. In CombCAT, for each query, we first group documents into different categories such that documents that appeared in n different rankings are put in the same category $category_n$. Then we proceed with our re-ranking by first ranking documents in decreasing order of n, and within $category_n$ in decreasing order of sum score. Thus CombCAT explicitly rewards documents that appear in the largest number of rankings.

For one set of runs, we use Bing to obtain "related queries" for each query. Then we use Indri to run each of these related queries as well as the original query. We then aggregate the resulting rankings using CombCAT. This produces our Session track **Udel14Run1** RL1 run as well as our Web track **UdelCombCAT** runs.

The **Udel14Run1** RL3 run takes advantage of the additional session data by collecting all of the last queries for all sessions on the same topic, collecting "related queries" for these, submitting these to our indri index, then aggregating results using CombCAT.

5.3 Results

Table 5 shows Session track results for nDCG@10, while Table 6 shows Web track results for ad hoc and diversity measures. The results show that, using the official measure nDCG@10, CombCAT

Runs	RL1	%change	RL3	%change
udel_itu	0.1515	0%	-	-
udel14Run1	0.2262	0%	0.2432	7.5%

Table 5: Session results (nDCG@10)

Run	nDCG@20	ERR@20	α-nDCG@20	ERR-IA@20	strec@20
udelCombCAT2	0.26135	0.17880	0.655802	0.583244	0.834286
udel_itu	0.19938	0.17536	0.565817	0.472367	0.831190
udel_itub	0.18607	0.16364	0.534793	0.443358	0.812524

Table 6: Graded relevance measures and diversity measures

with Bing related queries outperforms simple field and phrase features. Additionally, incorporating information from other user sessions successfully improves the performance.

However, as Table 7 shows, the field & phrase features are less "risky" than CombCAT. When poorly-performing queries are discounted at a weight of $\alpha = 5$, the two field-and-phrase runs significantly outperform the CombCAT run. Interestingly, ignoring the full page and just using title, inlink text, and url (the **udel_itu** run) is slightly less risky than using the full page.

6 Conclusion

The Information Retrieval Lab at the University of Delaware participated in five tracks: Contextual Suggestion, Federated Web, Microblog, Web, and Sessions. In all five we performed well, with many of our runs above the average median, and in many individual topics/scenarios achieving the best performance.

References

[1] A. Bah, P. Chandar, K. Sabhnani, M. Zengin, and B. Carterette. University of Delaware at TREC 2013. In *Proceedings of TREC*, 2013.

[2] L. Bednarik and L. Kovács. Efficiency analysis of quality threshold clustering algorithms. *Production Systems and Information Engineering*, 6, 2013.

[3] G. V. Cormack, M. D. Smucker, and C. L. A. Clarke. Efficient and effective spam filtering and re-ranking for large web datasets. *Information retrieval*, 14(5):441–465, 2011.

[4] E. Kanoulas, B. Carterette, P. D. Clough, and M. Sanderson. Overview of the TREC 2012 Session track. In *Proceedings of TREC*, 2012.

[5] D. Nguyen, T. Demeester, D. Trieschnigg, and D. Hiemstra. Federated search in the wild: the combined power of over a hundred search engines. In *Proc. CIKM*, 2012.

[6] T. Strohman, D. Metzler, H. Turtle, and W. B. Croft. Indri: a language model-based search engine for complex queries. *ICIA*, 2005.

	rm (indri)	terrier	udel_itu	udel_itub
α=0	0.02584	-0.00992	0.00344	0.01516
α=5	-0.11092	-0.27713	-0.37582	-0.32152

Table 7: ERR@20 Risk-aware measures for udelCombCAT2 relative to the other runs (listed in columns)