HU_DB at TREC 2014 Microblog Track

Jennifer Klein, Yishai Oltchik, Nerya Or, Sara Cohen

The Rachel and Selim Benin School of Computer Science and Engineering,
The Hebrew University of Jerusalem

***Abstract*—This paper describes our system for the Tweet Timeline Generation (TTG) task of the Microblog track, at the Text Retrieval Conference (TREC) 2014. Intuitively, given a collection of microblog posts (i.e., tweets), and a keyword query Q, the goal is to generate a timeline of related tweets. Such a timeline consists of representative tweets, relevant to Q. In our system we employ query expansion to identify highly relevant tweets, and then use affinity propagation to cluster the tweets, based on their word similarity, hashtag similarity and temporal similarity. We then return a representative tweet from each cluster. The result is a system with relatively good precision, but, unfortunately, poor recall. We discuss the techniques employed, as well as the insights gleaned while developing and testing our system.**

I. Introduction

Microblogging platforms, such as Twitter, Google+ and Tumblr, are a popular means to share and discover timely information. Such platforms differ from standard social networks, in that users post only short messages. As such, microblogging gives rise to new information retrieval challenges. One such important challenge, which is the focus of this TREC task, is that of summarizing non-redundant information, relevant to a given information need. Both the problem of finding relevant microblog posts, and that of grouping them into disjoint clusters, is challenging, due to the brevity of microblog posts.

The Microblog track at TREC 2014 focused only on Twitter, and defined the following task, called the *Tweet Timeline Generation (TTG) task*:

Task 1. *Given a keyword query Q and a time cutoff t, retrieve a* timeline *of the most informative tweets about Q which were posted no later than t, where a timeline is simply a list of representative tweets, sorted according to time.*

As part of the TREC framework, an online API to a historical Twitter dataset was made available. Keyword queries were dispatched to the API and matching sets of tweets were returned. We used only this API in our implementation, and no additional external datasets. Our goal was to determine how well queries can be answered in this fashion, i.e., whether one must resort to additional external data in order to achieve satisfactory precision and recall in such a setting.

The TTG task poses several inherent difficulties. First, tweets are very short. Hence, determining whether a tweet is relevant is a difficult task. Unlike longer documents, considered in standard information retrieval tasks, there are few context clues that can help determine the tweet's topic focus. Second, as we are returning representative tweets, each such tweet should be non-redundant, given the other tweets, i.e., tweets should each contain distinct information one from another, and thus, every tweet containing distinctly different information should be returned. Third, the system should not return too many tweets, as the goal is to create a timeline of representative related tweets, and not to exhaustively return every related tweet. Thus, tweets similar one to another should have a single representative in the final results.

We note that the second and third challenges give rise to rather contradictory solutions. In order to make sure that all representatives are returned (Challenge 2) one is motivated to return many tweets. However, too many representatives should not be returned (Challenge 3), and thus, among tweets that are similar (but not identical) a single representative should be chosen. To deal with these challenges, we clustered the tweets and returned a single representative from each cluster. As it turned out, choosing the correct cluster sizes (and thereby determining the number of representatives) is critical to achieving satisfactory recall. As we discuss later on, our choice of too large clusters seems to have been a determining factor which caused our system to have recall that was insufficient.

This paper is organized as follows. Section II describes our system. Empirical results of our submitted runs are presented in Section III. In Section IV we discuss insights gleaned from participation in this task. Finally, in Section V we conclude.

II. System Overview

Given a keyword query Q and time t, our system proceeds in three stages:

1) Find relevant tweets, for the given query.
2) Cluster the tweets into highly related semantic groups.
3) Choose a representative tweet from each cluster.

Finally, we return a timeline consisting of a time-sorted list of representative tweets. Each of these stages is discussed in detail below.

A. Generating a Corpus of Relevant Tweets

In the first stage, the system gathers tweets that seem to be relevant to the given keyword query Q. To find such tweets, we start by removing stop words from Q, thereby deriving a new query Q_1. We dispatch Q_1 to the TREC API as a conjunctive query. If Q_1 returned no results, we attempt to dispatch variations of Q in which one or more terms is omitted. This gives us a first set of relevant tweets R_1.

We then perform query expansion, to derive additional, relevant, tweets as follows. Let W be the set of words w in R_1 that satisfy the following conditions:

- w is not a stop word;
- w appears in at least $\alpha|R_1|$ of the tweets of R_1.

Intuitively, w is a popular term in the results of Q_1, and is not a stop word, and hence, is likely to be relevant to query Q. After some testing, we chose $\alpha = 0.2$.

Now, if query Q_1 contained at least i terms, we create and dispatch additional queries, using the words in W. These new queries are created in the following fashion. For each word w_1 in Q_1 and each word $w \in W$, we create a query $Q_{w_1 \leftrightarrow w}$ derived by swapping w_1 with w. Note that a choice of a sufficiently large i is necessary to avoid topic drift. In our experiments, we often (but not always) chose $i = 4$. Additional details appear in Section III.

Let R_2 be the set of tweets containing both R_1, as well as the results derived by all queries created in the expansion process. According the the TREC guidelines, we removed from R_2 tweets that are deemed to be "retweets" (i.e., duplicates of other tweets, generated as a result of a user sharing a tweet generated by another user), as well as non-English tweets. We also removed duplicates tweets. Note that we removed illegal tweets from R_2 (and not from R_1), as such tweets can contain valuable information (such as frequent words), even if they should not be returned to the user. Let R be the set of remaining tweets. We use R in the next stages, to create our clusters and final results.

B. Clustering the Tweets

In the second stage, we cluster the corpus of tweets R, which was obtained in the first stage. The goal is to place similar tweets in the same cluster. However, determining whether tweets are similar is not a trivial problem. Our system is based on three assumptions:

- *Tweets containing similar sets of words are similar.* This is natural, as tweets using mostly the same words are likely to be on the same topic.
- *Tweets containing similar hashtags are similar.* Hashtags are essentially labels, chosen by the microblog poster, to describe his or her tweet. If users choose the same labels, their tweets are likely to be on the same topic.
- *Tweets that were tweeted close in time one to another are similar.* Since many tweets are tweeted based on breaking events, if users post tweets at similar times, their tweets are likely to be in reaction to the same event.

We now consider how to turn each of these assumptions into a numerical similarity score.

Let t_1 and t_2 be tweets.

Definition 1 (Word Similarity)**.** *Let $\mathcal{B}(t_i)$ be the bag of words derived by* (1) *removing stop words from t_i and* (2) *stemming all remaining words. Then, the* word similarity *of t_1 and t_2, denoted $d_{word}(t_1, t_2)$, is the normalized Dice coefficient over $\mathcal{B}(t_1)$ and $\mathcal{B}(t_2)$, i.e.,*

$$d_{word}(t_1, t_2) = \frac{|\mathcal{B}(t_1) \cup \mathcal{B}(t_2)| - 2|\mathcal{B}(t_1) \cap \mathcal{B}(t_2)|}{|\mathcal{B}(t_1) \cup \mathcal{B}(t_2)| - |\mathcal{B}(t_1) \cap \mathcal{B}(t_2)|} .$$

Note that bag union and intersection is used in the above formula.

Definition 2 (Hashtag Similarity)**.** *Let $\mathcal{H}(t_i)$ be the set of hashtags in t_i. Then, the* hashtag similarity *of t_1 and t_2, denoted $d_{hash}(t_1, t_2)$, is the Jaccard coefficient of $\mathcal{H}(t_1)$ and $\mathcal{H}(t_2)$, i.e.,*

$$d_{hash}(t_1, t_2) = 1 - \frac{|\mathcal{H}(t_1) \cap \mathcal{H}(t_2)|}{|\mathcal{H}(t_1) \cup \mathcal{H}(t_2)|} .$$

Finally, we define the temporal similarity of tweets t_1 and t_2.

Definition 3 (Temporal Similarity)**.** *Let m be the maximal time difference between the timestamps of any two tweets in R. Let $ts(t_i)$ be the timestamp of t_i. Then, the* temporal similarity *of t_1 and t_2, denoted $d_{time}(t_1, t_2)$ is defined as follows*

$$d_{time}(t_1, t_2) = 1 - g_m(|ts(t_1) - ts(t_2)|) ,$$

where g_m is the Gaussian bell curve formula where parameter σ is chosen to be a linear function of m.

In order to compute the distance between t_1 and t_2, we used a linear combination of the three similarity measures defined above. We considered three weighting schemes:

- *Equal Weight:* In this weighting scheme, we gave equal weight to all three parameters, based on the intuition that a priori, it is not known which is the most important, i.e.,

$$d_{eqWeight}(t_1, t_2) = \frac{d_{word}(t_1, t_2) + d_{hash}(t_1, t_2) + d_{time}(t_1, t_2)}{3} .$$

- *Decreased Temporal Weight:* Due to the large number of tweets that are posted at any given moment in time, it is likely that many unrelated tweets will be posted at the same time. Hence, in this scheme, we give lower relative weight to temporal similarity, defining,

$$d_{lowTemporal}(t_1, t_2) = \frac{5\left(d_{word}(t_1, t_2) + d_{hash}(t_1, t_2)\right) + 2 \cdot d_{time}(t_1, t_2)}{12} .$$

- *No Temporal Scoring:* Finally, in our last weighting scheme, we completely ignored temporal similarity, as this factor may sometimes be more misleading than helpful, and defined

$$d_{noTemporal}(t_1, t_2) = \frac{d_{word}(t_1, t_2) + d_{hash}(t_1, t_2)}{2} .$$

Now, given a tweet distance function d (which is one of $d_{eqWeight}$, $d_{lowTemporal}$, or $d_{noTemporal}$), we create a distance matrix M^d, with dimensions $|R| \times |R|$. The entry in place $M^d[i, j]$ is simply $d(t_i, t_j)$, where t_i and t_j are the i-th and j-th tweets in R. Note that identical tweets will have distance 0, and the largest possible distance value is 1.

Finally, we cluster the tweets using M^d with the Affinity Propagation clustering algorithm [1]. Affinity Propagation is an iterative algorithm that uses message passing between samples in the data set in order to determine which samples are best exemplified by which other samples. One of the significant advantage of Affinity Propagation over other, more standard, clustering algorithms, is that the number of clusters does not have to be predetermined. This was a distinct advantage in the context of the TREC TTG task.

C. Cluster Representative Selection

In our third, and final stage, for each cluster C created in the second stage, we choose a representative tweet. This is necessary in order to generate the final desired result—a timeline of informative tweets.

Let C be a cluster of tweets. Two ideas were used in order to choose a representative for C:

- *Centrality:* Intuitively, the closer a tweet is to the entire cluster, the more similar it is to all tweets in the cluster. Formally, for $t \in C$, and a tweet distance function d, we define
$$cent_d(t, C) = \frac{1}{\sum_{t' \in C} d(t, t')} .$$
- *Amount of retweets:* Tweets which were shared many times by other users were deemed highly relevant by real users. Hence, such tweets received a boost in ranking. Formally, we define $retweet(t)$ as the number of times that t was retweeted.

For a representative tweet $rep(C)$ of C, we chose the tweet t in C that maximized the combined centrality and retweet scores

$$rep(C) = \arg\max_{t \in C} (cent_d(t, C) + retweet(t)) .$$

In our implementation, we also considered different ways of combining these two factors, with similar results derived.

III. Results

A total of 4 runs were submitted by our team, denoted *Standard, SR, SRTD* and *SRTL*. These runs differed one from each other in our choice of i (the minimal size of queries for which expansion is performed) and d (the tweet distance function). These choices appear in Table I.

Run Name	i	d
Standard	3	$d_{eqWeight}$
SR	4	$d_{eqWeight}$
SRTD	4	$d_{lowTemporal}$
SRTL	4	$d_{noTemporal}$

TABLE I
Run parameters.

TTG task outputs are assessed according to two metrics: precision and recall. Precision is measured as the ratio of semantic clusters represented, divided by the total size of the result set. Recall is defined as the ratio between the number of semantic clusters represented and the actual number of semantic clusters that exist in the dataset, as determined by a human assessor. Additionally, a weighted variant of scoring was used, in which tweets that were marked as highly relevant by the human assessors contributed twice as much weight to the score of results that contained them.

Fig. 1. Unweighted recall vs. precision of all submitted TREC microblog runs.

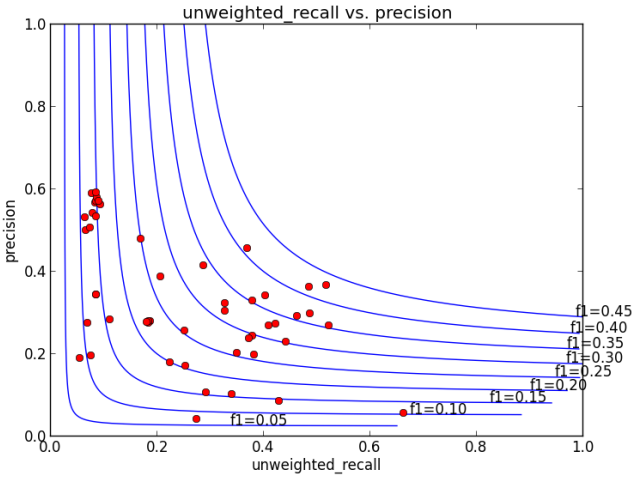

Fig. 2. Weighted recall vs. precision of all submitted TREC microblog runs.

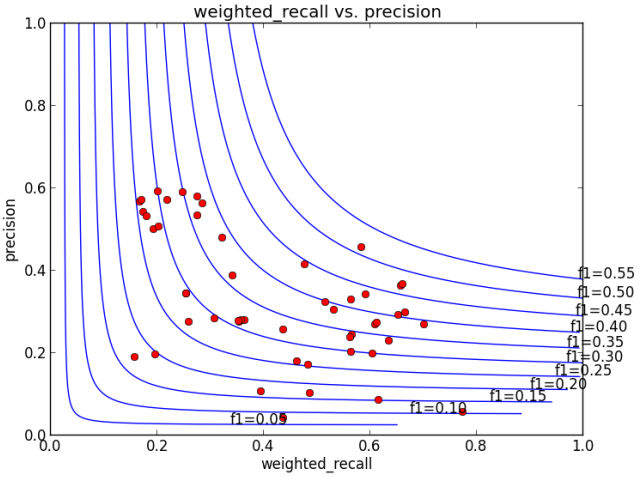

Table II shows the results of our submitted runs. Figures 1 and 2 are scatter-plots of the results of all runs submitted by teams participating in the 2014 TREC microblog track. Within these figures, our results form a cluster around the top-left area, displaying high precision but relatively low recall performance. The low recall reflects the fact that our system tends to partition the tweet corpus into fewer large clusters, rather than many small ones.

IV. Limitations and Suggested Improvements

The TTG task was new for 2014, and thus, this was our first time in participating in such an initiative. In this

Run	Unweighted Recall	Weighted Recall	Precision
SR	0.0780	0.2481	0.5905
SRTD	0.0868	0.2764	0.5798
SRTL	0.0942	0.2851	0.5615
Standard	0.0850	0.2759	0.5337

TABLE II
RESULTS OF OUR SUBMITTED RUNS: UNWEIGHTED/WEIGHTED RECALL, AND PRECISION.

section we discuss insights, system limitations and suggested improvements, gleaned in a post-mortem, from our system results. It is our hope that these will be useful for others, interested in solving the TTG task (or similar problems) in the future.

Fixed parameter values. Our choices of i and d were fixed within a run. However, in practice, it is apparent that different choices were preferable for different queries. More experimentation is needed to learn query properties that imply preference for one set of parameters over another. If such learning is accomplished, a single run could combine different parameters for different queries, thus improving their results, in general.

Going beyond stop-words. Our system used a coarse separation of query terms into stop-words and non-stop-words. In practice, finer distinctions were necessary to determine words that are critical to a query, and should never be removed (i.e., swapped out) in the query expansion process. For example in the query 'Argo wins Oscar', the term 'Argo' is critical and it is virtually impossible to obtain a good set of results when searching without this term. The term 'wins' is not a stop-word, yet could have been omitted without incurring too much query drift.

Grammar-based term modification during query expansion. The corpus of tweets against which we applied queries was available only via the API. Hence, we could not perform stemming on this corpus. Thus, queries using terms in a different grammatical form from that of a tweet, could not return the tweet. A possible solution to this problem would have been expanding the query with modified grammatical forms of query terms, e.g., for the query 'injuries by pets', one could expand with terms such as 'injury' and 'pet'.

Using clustering to find representative tweets. Our system was built leveraging the idea that by clustering tweets, and choosing cluster representatives, one can create a timeline of representative tweets. Our clustering technique created relatively large clusters. At development time, this was deemed to be an advantage to the system, i.e., that it succinctly summarizes large numbers of tweets. In practice, our approach towards choosing representatives did not agree with that of the human assessors (who had many more representatives than us), and thus, our system received very low recall scores. We note that the semantic clusters created by the human assessors were often singletons, and thereby giving clear preference by them to large numbers of representative tweets.

Ontologies or external knowledge bases. Our system did not employ any sort of ontology or external knowledge base in order to better perform query expansion. As it turned out, this was a bad design decision. As tweets are short, it is not possible to find all related terms only by taking popular terms within the first tweet set R_1. Use of an external knowledge base or ontology would have been helpful in overcoming this difficulty, e.g., in queries such as "injuries by pets', expanding pets with dogs, cats, etc., using an ontology, would have been beneficial.

Query expansion using hashtags. In Twitter, it is common that an event has more than one hashtag associated with it. Thus, by determining which hashtags are related to an event, one can derive new relevant queries to run and obtain more data. Thus, query expansion should proceed not only with words, but also with hashtags. This was not included in our system.

Ambiguous queries. Choosing relevant tweets and creating timelines was made more difficult due the fact that (as often happens in real life), queries were ambiguous, and their interpretation as information needs were not available during system development. Some queries were interpreted in a very narrow fashion by the human assessors, e.g., 'cherry blossom Washington' was interpreted as tweets expressing interest in attending the National Cherry Blossom Festival in Washington (and not information about cherries blossoming in Washington). Obviously, query interpretation has a significant effect on precision, but it is difficult to overcome this issue in an information retrieval system.

In addition, the TTG task seems to be well fitted for queries searching for information about events, as in such a case it is natural for a timeline to be generated. On the other hand, queries looking for specific information do not seem to be suited for TTG. Some of the queries given for the task fell into the second category, which made finding representative tweets, in a timeline, more difficult. The ability to automatically differentiate between such queries, and apply different techniques, depending on query type, would be a

useful future system feature.

V. Conclusion

Our system employed a fairly simple mechanism, based on query expansion and clustering, to solve the TTG task. Even though external data sources were not used, relatively high precision was achieved, with somewhat less satisfactory recall. Besides providing system details, we have discussed new directions and extensions that can help to significantly boost the recall. We leave such implementations for future work.

VI. Acknowledgments

The authors were partially supported by the Israel Science Foundation (Grant 1467/13) and the Ministry of Science and Technology (Grant 3-9617).

References

[1] B. J. Frey and D. Dueck. Clustering by passing messages between data points. *Science*, 315:972–976, 2007.