MPII at the TREC Microblog Track 2015

Kai Hui[1,2] and Klaus Berberich[1]

[1] Max Planck Institute for Informatics, Saarbrücken, Germany,
[2] Saarbrücken Graduate School of Computer Science
{khui, kberberi}@mpi-inf.mpg.de

Abstract. In this technical report, we describe our contribution to the TREC Microblog Track 2015. We developed an online system for both the mobile notification and email digest tasks. Our code is made publicly available on GitHub.
Our system employs Apache Lucene to cache the incoming tweets, generating results for both tasks by periodically retrieving tweets from the cache, and afterward using prediction and learning-to-rank algorithms to rank tweets.

1 Introduction

Microblogs, like Twitter, nowadays become an information hub for our daily lifes, apart from its social network function. Users keep creating, forwarding, and receiving all types of information, to spread and absorb information, e.g., breaking news, professional materials, and casual stories. Huge volumes of tweets dynamically appear and spread everyday. Therefore automatic methods to facilitate users in filtering information are desired. Inspired by this type of information need, the TREC Microblog Track 2015 introduces two separate tasks, namely the mobile notification task and the email digest task, requiring systems to filter and feed users with interesting tweets in a personalized manner on a daily basis. The mobile notification task simulates the scenario where users may read recommended tweets on their mobile device, emphasizing low latency; The email digest task, on the other hand, targets another user case by pushing tweets via email at the end of each day. In the evaluation, both tasks consider how well selected tweets match user profiles, i.e. their relevance to the specific user, and penalize redundancy, thereby encouraging novelty of results. The mobile notification task emphasizes the system's real-time ability, considering latency in the evaluation.

The Microblog track dates back to2011 and has been held for four consecutive years. Previous tasks were shaped as retrieval tasks, requesting systems to retrieve relevant tweets given the user's information need. However, inherent properties of tweets, such as their restricted length, rich meta data, and temporal sensitivity distinguish them from traditional web search. Despite the first introduction of the tasks in this year, there is common ground shared with past tasks. Given that user profiles are in the same format as queries in previous years, the email digest task is actually a typical retrieval task on microblogs,

considering search result novelty, and using a similar evaluation strategy as in the tweet timeline generation (TTG) task from 2014. Additionally, the target scenario in the mobile notification task is closely related to that of the filtering task from 2012, selecting relevant tweets for a specific user.

Nevertheless, several crucial differences distinguish this year's tasks, raising particular difficulties.

1. **Live stream:** Instead of relying on simulation based on archived tweets, this year's tasks are directly conducted on the twitter stream, thus requiring the ability to process the large volume of tweets in the twitter live stream in a real-time manner;
2. **Novelty:** The evaluation for both tasks considers the novelty, leading to decision making beyond an independent consideration of the individual tweet;
3. **Emphasis on real-time:** In the mobile notification task, the latency between the tweet being posted and the notification will be considered in the evaluation, and a latency of more than 100 minutes will lead to no effect on the effectiveness score, regardless of the relevance of the returned tweet.

Our group implemented systems for both tasks and submitted three runs for each of them [3]. Overall, in response to the real-time requirements as well as the large volume of tweets in the live stream, we firstly cache the incoming tweets in an Apache Lucene index and select candidate tweets for fine-grained processing by ranking cached tweets periodically. Afterwards, we predict the relevance independently for each tweet from the short list of top-ranked tweets, using different strategies to make "mobile notifications"; meanwhile we sort tweets at the end of each day with different algorithms to generate an "email digest". The different runs submitted result from combinations of different prediction methods and different decision making strategies.

In Section 2, we highlight some features of our systems. Their detailed design is described in Sections 3 and 4. Finally, in Section 6, we will give a summary of our work.

2 Overall Design

As mentioned before, the three main issues to be addressed by the systems are as follows:

1. **Real-time:** The arrival rate of tweets is about 30 tweets per second, requiring the systems to process them fast enough and discard most tweets after an initial pre-processing, only storing their statistics in favor of prediction;
2. **Effectiveness and Low Latency:** These two objectives conflict with each other, low latency means demands that we make a decision upon arrival of individual tweets. Meanwhile we may want to wait to see whether more (possibly more) relevant tweets arrive in the future;

[3] https://github.com/lhyan792/microblogtrack

3. **Novelty:** Ideally we want to keep track of all decisions made, avoiding to push duplicate tweets to a specific users. This, however, may reduce system efficiency over time as more and more tweets have been pushed as notifications.

Addressing the aforementioned challenges, our system is designed with the following features:

1. **Apache Lucene Cache:** We index all English tweets and keep retrieving top tweets for further fine-grained processing, resulting in fewer tweets to focus on;
2. **Latency vs. Effectiveness:** The periodicity of the retrieval together with the depth of the top results retrieved govern this balance: longer duration leads to better relevance but longer latency, meanwhile inspection of more candidate tweets leads to more diverse but less relevant tweets. We will discuss this in detail in Section 3.2;
3. **Caching of Results:** We can tune the number of tweets to compare against by only tracking the results from the last few days, considering the dynamics of the twitter live stream.

Our system includes four components: the **listener**, the **filter**, the **predictor**, and the **decision makers**. The listener tracks the live stream and stores all English tweets in an Apache Lucene index, accumulating global statistics for the live stream; the filter periodically retrieves most relevant tweets for each user profile for further processing; afterwards, a pointwise predictor estimates the relevance of individual tweets independently and the listwise predictor further sorts the tweets to generate the final rankings; finally, the decision maker decides based on the prediction results and outputs results for both tasks periodically. More specifically, we employ the open source Hosebird Client (hbc) [4] from twitter as our listener; employ Apache Lucene's near-real-time search features to power our real-time filter; sequentially, the retrieval scores and logistic regression are used as two separate pointwise predictors, and learning-to.rank algorithm MART [4] and MAXREP algorithm [5] are used for listwise prediction. For both predictors, a semantic match score from Apache Lucene together with different tweet meta-data are used as features. In decision making, we use the distribution of prediction scores and handcraft rules to make final decisions.

Our submitted runs are summarized in Table 1, and the details of each method are described in Section 3.

3 Implementation Details

As mentioned before, our systems consist of four components, which act in sequence: the listener, the filter, the predictor, and the decision maker. In this section, we describe each of them in detail.

[4] https://github.com/twitter/hbc

Table 1. Details of submitted runs

Task	Run Name	Description
Mobile Notification	MPII_LUC	Averaging Lucene retrieve scores
	MPII_COMB	Combination of MPII_LUC and logistic regression
	MPII_HYBRID	Intersection of MPII_LUC and MPII_COMB
Email Digest	MPII_COMB_MART	MART based on MPII_COMB
	MPII_LUC_MART	MART based on MPII_LUC
	MPII_COMB_MAXREP	MAXREP based on MPII_COMB

3.1 Listener

The open source Hosebird Client from twitter is used to track the live stream during the experiment period. We received around 30 tweets per second during this time. Only English tweets are retained by using the language tag from twitter, excluding all non-English tweets. That is, tweets lang-tag not equal to "en" are discarded. All remaining tweets are stored in an Apache Lucene index. The details of the Apache Lucene index are described in next section.

3.2 Filter

After removing non-English tweets, volume of tweets is still too large for online processing, and the filter component is used to further shrink reduce it, thereby focusing on more promising candidate tweets.

In particular, the open source retrieval system Apache Lucene[5] is employed to periodically retrieve the most relevant tweets for each user profile. We keep writing received tweets into the Apache Lucene index, together with their arrival timestamps. In every t minutes, we fetch the top-k among recently received tweets as an input for follow-up components. More specifically, the title, the description, and narrative fields from the given user profile are used as separate queries. Moreover, we employed Rocchio's query expansion[6] to select the top-10 expansion terms based on Wikipedia dump[7] as a query. Henceforth, for each user profile, there are four query variants. We also use four state-of-the-art retrieval models integrated into Apache Lucene to weight each user-profile tweet pair, specifically TfIdf [6], Okapi BM25 [6], a unigram language model with Dirichlet smoothing [8], and the divergence from randomness model (DFR) [1] with inverse term frequency as a basic model and Laplace normalization as first normalization of information gain. Thus, in total four query variants and four retrieval models are employed on each user-profile tweet pair, leading to sixteen different retrieval scores. The union of top-k tweets from all these combinations are fed to follow-up procedures. As mentioned in Section 2, the choices of parameters t and k

[5] https://lucene.apache.org/
[6] http://lucene-qe.sourceforge.net/
[7] https://dumps.wikimedia.org/

include a trade-off between recall, i.e., the coverage of relevant tweets, and the latency of the mobile notification, i.e., how soon our system can push the most recent arriving tweets to users. Larger t and k lead to better recall: with both t and k large enough, all tweets will be processed in the follow-up components, at the cost of high latency; comparably, smaller t and k lead to fewer tweets for fine-grained processing and to lower latency between the arrival of tweets and the decision making. In our configuration, t is set as 15 minutes and k is set to 10 for both tasks. Considering the different evaluation benchmarks in the evaluation, however, the latency actually played no role in the email digest task. Thus, ideally, we should employ different strategies, i.e., repeatedly retrieving tweets after a short period of time for the mobile notification task and retrieving tweets only once per day for the email digest. We may implement this feature in the future.

3.3 Predictor

After filtering, more expensive algorithms are affordable. On average, only several thousand tweets are retained per day, on which we employ different pointwise and listwise prediction methods. We introduce different prediction methods for both tasks in this section, and thereafter in Section 4, we separately introduce the features used by both predictors.

Pointwise Predictor For pointwise prediction, sixteen retrieval scores from Apache Lucene, together with different tweet meta-data are summarized to infer the relevance of individual tweets w.r.t. to a specific user profile. The results from this pointwise prediction are used in decision making for both tasks.

One straightforward strategy would be to directly combine the different retrieval scores. Beyond that, we also use logistic regression to further include different tweet meta-data, using the probability output by the model as a prediction score.

- **Linear combination of relevant weightings from Lucene** (Luc): averaging the second to seventh highest retrieval score from different retrieval models individual tweets, where the partial inclusion of the sixteen retrieval scores is to improve the robustness;
- **Logistic regression:** to further include the tweet meta-data, logistic regression (Liblinear [3] is used) and trained on labeled data from the past four iterations of the TREC Microblog Track;
- **Linear combination of above scores** (Comb): inspired by the observed high false positive and high true negative rate in an offline test, which indicates a too "optimistic" prediction from logistic regression, we further design a joint model, aiming at a more conservative prediction. For each tweet-query pair with a relevance probability larger than 0.5 from logistic regression, the score is linearly combined with the retrieval score described in the first method, where the probability from logistic regression is weighted 0.3;

– **Hybrid method** (HYBRID): intersecting the first and the third methods as results. Due to the fact that the evaluation measures actually penalize verbose outputs, this method creates the most conservative prediction. On average, for each query there were only three tweets being returned under this configuration.

Listwise predictior The listwise predictor is used for the email digest task. Different from the pointwise predictor, it only considers the relevance and novelty, sorting all tweets retrieved within one day according to the listwise score after excluding all duplicate tweets. More precisely, we track tweets passed by the filter component and retain the top-n tweets with highest pointwise score.

– **Multiple Additive Regression Trees [4]** (MART): Given the fact that the feature design influences the performance more than the model choice, we select this simple yet effective pointwise learning-to-rank method in our submission. We configure to re-rank on a relative deeper search results by setting $n = 800$;
– **Maximum Representation [5]** (MAXREP): selects the most representative documents from a large set of candidates. This method fits in the email digest task in the sense that it is able to pick a set of diverse tweets. After selecting the most diverse 100 tweets, we re-rank the tweets according to the pointwise score as final output; considering that the MAXREP algorithm does not consider the relevance of the tweets, we apply it to a relatively shallow candidate pool with $n = 300$.

3.4 Decision Maker

At this stage, we make decisions to generate final results for both tasks. For the email digest task, we simply output top-100 tweets from the listwise prediction at the end of each day. Therefore, we only describe the method for the decision maker in the mobile notification task. For both decision makers, we keep recording tweets we pushed, and we compare each new tweet against this set using the Levenshtein edit distance as a similarity measures to exclude near-duplicate tweets.

Given that latency will be penalized in the mobile notification task, we desire to push relevant tweets upon their arrival. However, as mentioned before, these two targets actually conflict with each other, especially considering that we are only allowed to push at most 10 tweets per day. In fact, since we have no clue about future tweets when we make individual decision, resulting in a "local optimized" decision in the sense that future tweets could be better. To fix this, we employ similar strategy as in the secretary problem [8]. In the secretary problem, irreversible decisions are made upon the arrival of each candidate, aiming at maximizing the probability of picking out the best candidate. The idea is to skip certain first n candidates, afterwards selecting the first candidate that is

[8] https://en.wikipedia.org/wiki/Secretary_problem

Table 2. Summary of features

Category	Feature Description
Semantic Scores	Sixteen retrieve scores from Lucene
User Authority	Number of followers Length of the user description Number of the urls in the user description Number of tweets the user marked as favorite Number of friends Whether user is identified as a celebrity Whether user uses default image as his (her) icon Number of lists the user gets involved in Number of posted tweets
Tweet Quality	Number of favorites Number of Hashtag contained Number of times being retweeted Number of embedding urls Number of other users being mentioned in the tweet Length of the tweet

better than the best of first n candidates. Likewise, we skip ax certain number of tweets at the start of each day, and start to make decision thereafter. We use the accumulated pointwise prediction score to convert absolute prediction scores to a relative score, suggesting how good the tweet is in comparison to the already-observed ones. In particular, we wait for 150 minutes, i.e. 10 periods of filtering, before making a decision on each day. Afterwards, for an individual tweet, a threshold of the absolute pointwise prediction score is used to decide whether to push the tweet. For the first tweet in each day, since we have no absolute score threshold, we employ three handcrafted rules instead: 1) the pointwise prediction score should be among the top 0.1 percent; 2) the tweet is long enough; and 3) if it contains an embedded URL, the similarity between the URL's title and the query title field should be high enough. After the first tweet is selected, its absolute pointwise prediction score is used as the threshold, and dynamically adjusted by 1% after each acceptance or rejection, until 10 tweets have been pushed.

4 Features

In this section, we describe the features used in all aforementioned learning algorithms. Different features shown to be effective in existing literature [2] [7] are summarized in Table 2. Features can be categorized into three groups: semantic matching features, user authority features, and tweet quality features. Intuitively, semantic features imply how relevant the tweet is w.r.t. to the given user profile, user authority features indicate how likely the tweets from the particular user are of interest to the other user in the past; and the tweet quality features

Table 3. Evaluation results of submitted runs

Task	Run Name	Evaluation Results
Mobile Notification	MPII_LUC	ELG: 0.0841, nCG: 0.1700
	MPII_COMB	ELG: 0.0575, nCG: 0.1104
	MPII_HYBRID	ELG: 0.1025, nCG: 0.0777
Email Digest	MPII_COMB_MART	nDCG: 0.0275
	MPII_LUC_MART	nDCG: 0.0310
	MPII_COMB_MAXREP	nDCG: 0.2093

indicate how likely the tweet is informative in general, e.g., how likely it is of being retweeted, the more likely the tweets are informative etc.. Additionally, for semantic features, since the scale of the retrieval scores from different models vary a lot, e.g., from less than one to hundreds, it makes no sense to directly use this absolute score. Therefore, we keep track of the maximum and minimum scores for each of them, and conduct min-max normalization before further computation.

5 Evaluation Result

The evaluation results are summarized in Table 3. For the mobile notification task, the most conservative method MPII_HYBRID achieves highest ELG as expected. However, comparing with MPII_LUC, the MPII_COMB performs worse, which indicates the prediction results from our learning model actually harmed the performance. The same thing happened in our email digest runs, the MPII_COMB_MAXREP performs much better than the other two, with the shallowest initial document pool.

6 Conclusion

We implemented systems for both the mobile notification task and the email digest task. Apache Lucene is employed to cache and select the most relevant tweets for further processing, making our system focus on a relatively small set of tweets. We use relevance weighting from Apache Lucene and different meta data from each tweet to predict its relevance to a specific user profile. In the future, we may further refine the features to improve effectiveness, and separate the systems for email digest in favor of the quality of the initial set of tweets.

References

1. G. Amati and C. J. Van Rijsbergen. Probabilistic models of information retrieval based on measuring the divergence from randomness. *ACM Transactions on Information Systems (TOIS)*, 20(4):357–389, 2002.

2. Y. Duan, L. Jiang, T. Qin, M. Zhou, and H.-Y. Shum. An empirical study on learning to rank of tweets. In *Proceedings of the 23rd International Conference on Computational Linguistics*, pages 295–303. Association for Computational Linguistics, 2010.
3. R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008.
4. J. H. Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.
5. K. Hui and K. Berberich. Selective labeling and incomplete label mitigation for low-cost evaluation. In *String Processing and Information Retrieval*, pages 137–148. Springer, 2015.
6. C. D. Manning, P. Raghavan, H. Schütze, et al. *Introduction to information retrieval*, volume 1. Cambridge university press Cambridge, 2008.
7. C. L. F. F. R. Qiang and Y. F. J. Yang. Pkuicst at trec 2014 microblog track: Feature extraction for effective microblog search and adaptive clustering algorithms for ttg.
8. C. Zhai and J. Lafferty. A study of smoothing methods for language models applied to ad hoc information retrieval. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 334–342. ACM, 2001.