Real-time Topic Detection and Tracking in Microblog: Towards A Comprehensive Tweet Recommendation System

Karankumar Sabhnani and Ben Carterette
{karans,carteret}@udel.edu
Department of Computer and Information Sciences
University of Delaware, Newark, DE 19716

ABSTRACT

Topic Detection and Tracking (TDT) has long been an important area of IR research. With the increasing availability of fast live streams of data, there is an increasing need for recommendation systems that automate the process of tracking these live streams and generating personalized digests of information. Microblog services such as Twitter, being a rich and rapid source of mostly-textual data, provide an ideal platform for developing and evaluating real-time TDT and recommendation techniques.

In this paper, we present a comprehensive recommendation system that tracks a "live" Twitter stream to generate real-time recommendations. Such systems come with stringent requirements for high precision, low latency, and minimal redundancy. To decrease time latency, we invert the usual query→document retrieval model, instead using documents (tweets) to retrieve "profiles" formed around a user's information need. To minimize redundancy, we represent these profiles as a collection of clusters of terms, only emitting a tweet if it does not seem to match any cluster in the collection. We conduct experiments on a 10-day-long live stream consisting of approximately 14 million English tweets, the data used for the TREC 2015 Microblog track, demonstrating that our system not only performs competitively in an online setting with no training, but also achieves more than 20% improvement when trained on historical data.

1. INTRODUCTION

Social medial services like Twitter, Facebook, and Tumblr have exploded in popularity: for Twitter in particular, the number of "tweets" issued per day has grown from only 5,000 in 2007 to over 500 million in 2013. One of the reason services like Twitter are popular is that they allow users to "follow" other users, allowing them see what those users are saying without requiring a reciprocal relationship. A user can choose to follow personal friends, celebrities, politicians, scientists, critics, etc, mixing and matching to put together a feed of tweets that they enjoy reading.

But the volume of tweets is so high that, whether because they follow too many others to be able to read everything in their feed, or they lack time to read the entire feed, or the people they follow do not share a particular interest, it is inevitable that one will miss things that would be interesting to them. For many users, it would be helpful to have a secondary service that scans the full Twitter stream, filters it, and sends those tweets that they might find most interesting.

Such a system must have very high precision: users typically want to see only what the people they are following are saying, so the system should have very high confidence that the user would find its recommendations interesting. It should also have low redundancy, suppressing things the user has already seen in their own feed or were previously served by the system itself. And in some cases, it should have low latency, providing those tweets immediately as available (either directly to the user, or slotted into their feed at the appropriate time ordering).

In this paper we propose a system that meets those requirements. Inspired by work on Topic Detection and Tracking (TDT) and recommender systems, it creates and tracks "profiles" of user information needs, a component of which are groups of topically-related tweets. By matching tweets from the Twitter stream against these profiles and their clusters, it serves high-precision, low-redundancy information very quickly. The profiles and clusters are dynamically updated as time progresses to ensure that the matching tweets remain relevant and novel. We evaluate our system in both online and offline settings: the TREC 2015 Microblog track [16] enabled participants to test systems online against 10 days of the live Twitter feed, then save that data for future experimentation. Our online system performs well compared to other submissions, and in the offline setting we obtain statistically significant improvements of 20% or more.

This paper is organized as follows: in Section 2 we discuss related work on TDT, recommender systems, and Twitter search. In Section 3 we present the details of our system. Section 4 describes our experimental setup, and in Section 5 we give experimental results and analysis. We conclude in Section 6.

2. RELATED WORK

Topic Detection and Tracking (TDT) has long been an important area of IR research. Started with a pilot study in 1997, TDT was evaluated under the DARPA Translingual Information Detection, Extraction, and Summarization (TIDES) program for another 7 years until 2004 [3, 2]. TDT is an attempt to intelligently monitor text streams; it involves problems such as story segmentation (determining when one story ends and another begins), topic tracking (identifying which topics documents are relevant over time), first story detection (identifying the initial appearance of a topic in a stream), link detection (finding topical links between documents), and so on. Diverse corpora [8] generated by accumulating information from multiple sources such as newswire text; transcripts of CNN broadcasts; automatic speech-to-text systems converting television, radio and web broadcast news shows, etc. were used for evaluating TDT systems. Over the years TDT has been researched extensively [14, 26, 4, 19, 13].

This gradually led to improved techniques such as adaptive filtering. Adaptive filtering techniques [10, 22] also monitor text streams, learning dynamically over time to boost accuracy and better identify documents for the information need. These techniques have become a part of more prominent recommendation systems [1]. Recommendation systems have become extremely common in recent years, and are applied in a variety of applications such as e-commerce, social networks, movies, music, news, microblog, etc. And as the popularity of such services has increased, these systems have faced additional challenges such as redundancy of information [28] in the data streams.

Today, with the increasing availability of fast live streams of data, there is an increasing need for real-time recommendation systems that automate the process of tracking these live streams and generating personalized digests of information. Microblog services such as Twitter, being a rich and rapid source of mostly-textual data, provide an ideal platform for developing and evaluating real-time TDT and recommendation techniques. Thus the Microblog track started at TREC in 2011 [20] with the purpose of evaluating search methodologies in microblogging environments like Twitter. Microblogs are different from traditional web documents in that they are limited to 140 characters in length. This demanded retrieval techniques that could identify relevance in smaller text fragments. This track started with a pilot ad hoc task with the goal of finding most recent and relevant tweets for a given query specified at a point in time. It continued in 2012 with minor changes along with a new task known as the filtering task [23]. Filtering tasks were revived in 2012 because of this new emerging stream of information. Additional challenges such as increasing redundancy led to the Tweet Timeline Generation (TTG) task in 2014 [15]. This task supplemented the challenges of ad hoc retrieval with issues from TDT and multi-document summarization. However, real-time evaluation has proven to be difficult; these tasks have mostly been evaluated on offline datasets of tweets crawled from Twitter.

A considerable amount of research has been done on the problem of tweet recommendation in offline datasets [7, 27, 24]. Chen et al. [7] propose a tweet recommendation methodology based on collaborative filtering techniques by which they utilize additional signals such as a user's social relations and a publisher's authority. Uysal et al. [24] leverage retweet signals and a user's likelihood of retweeting tweets for generating personalized tweet rankings. Much of this work uses crawls of a user's activity on Twitter along with the users they follow to generate datasets for evaluating their methodologies.

Zhou et al. [29] also use collaborative filtering techniques for generating keyword tags for profiles. Based on these tags they prune the incoming stream to push timely updates to users on topics they wish to follow. They evaluate their system on datasets from two sources (Twitter and Netease) crawled between 2006 and 2011 to measure aspects of real-time recommendation such as recommendation accuracy, processing time, and scalability.

Experiments have been performed on more recent live streams, but at a smaller scale. Magdy et al. [17, 9] present a news portal that extracts the most popular articles for tracked topics. Another work by the same authors [18] presents an adaptive method for following dynamic topics on Twitter. Both these systems start with topics that are expanded with a rich set of hand tuned keywords. These human generated keywords are used by the classifier for judging relevance on the incoming tweets. Wang et al. [25] propose the use of hashtags for adaptive microblog crawling. They show that their adaptive algorithm identifies more relevant tweets as compared to a traditional pre-defined keyword classifier.

We perform experiments with similar intentions on similar data streams but at a larger scale. Our work is different in the sense that we present a completely automatic recommendation system that learns from already-available information on the web without any human intervention. This enables us to evaluate our system on a wider number of topics.

3. REAL-TIME MICROBLOG RECOMMENDATION SYSTEM

Our system comprises several modules in sequence to perform real-time Topic Detection and Tracking on Twitter's live stream. Each module is meant to handle one of the requirements: low latency, minimal redundancy and high precision efficiently. To avoid time latency, we invert the usual query $\rightarrow$ document model, instead indexing "profiles" formed around a user's information need/query, and query the index using incoming tweets (Section 3.1 below). Redundancy is handled by a dynamic clustering algorithm that detects and bundles redundant tweets in real time (Section 3.3.2). Finally, simple tf-idf scoring is used to match tweets with profiles, with scores tweaked to perform pseudo-phrase match between tweets and profiles to ensure high precision (Section 3.3).

3.1 Indexing profiles

A traditional index-based retrieval model for this task would require a continuous process of dynamically adding incoming tweets to an index in order to keep collection statistics such as idf up-to-date. With a heavy stream, this can be quite resource-intensive. Thus, we invert the usual model: we index profiles with Lucene before beginning to pull tweets, then query that index with incoming tweets to retrieve the best matching profile(s). The best-matching tweets are used to keep profiles up-to-date, while tweets that are not relevant to any profile are simply discarded. This facilitates real-time matching of tweets with profiles without any delay, while also keeping the profiles up-to-date with the stream.

TREC 2015 Microblog information needs are given as standard TREC-style topics, composed of a keyword query, a description, and a narrative for each topic representing the type of information the user is interested in. We use this information to generate a *profile* for each topic. Initially only the query and description are stored in the index, in fields called "Title" and "Info" respectively. Table 1 lists the fields in our index.

Recommender systems are prone to the so-called cold start problem [12, 11], that if they start "cold" with no prior in-

Field Names	Contents
ID	Topic ID
Title	Title query + hashtags with count ≥ 20
Info	Topic description + term vector
Doc	Text from top-2 ranked documents

Table 1: Field names and their contents in our Lucene index.

formation, they may be unable to recover from early errors. The cold start problem can hamper system performance early on until it stabilizes and has enough information to make accurate predictions. We address this problem by expanding our profiles with additional information. This expansion is done before the evaluation period with two sets of information already available over the web.

3.1.1 Expansion with retrieved documents

The first set of information used to expand profiles consists of top-ranked documents for the topic's query. Such documents could be taken from many different sources; we used two in particular: 1) the CustomSearch API of Google and 2) an Indri index of the ClueWeb12 collection. We query these sources with each profile's query to retrieve only the top 2 ranked documents. Text from these documents is parsed, stemmed, and stopped using Lucene's built-in classes and added to the profile index in the "Doc" field (see Table 1).

3.1.2 Expansion based on retrieved tweets

The second set of information is extracted directly from Twitter. We query the Twitter Search API with each profile's query to retrieve up to 450 tweets per profile. From each of these tweets, we extract three items:

1. tweet text,
2. any hashtags associated with the tweet, and
3. screennames and user mentions associated with the tweets.

The texts of selected tweets are concatenated and stored in the Lucene index in the "Info" field along with the topic description; the precise method by which tweets are selected to be stored here is described in Section 3.2 below. Hashtags are stored with the title query in the "Title" field, but only those that occur more than 20 times. Screennames are also stored in the Lucene index, but since we did not use them in the retrieval, they are not included in Table 1.

3.2 Clustering tweets for novelty

Identifying novelty is a key requirement for our system. We use clustering to create and maintain clusters of tweets that represent different aspects or facets of the information need. By only pushing tweets that seem to not match these clusters, we aim to minimize redundancy.

We use the *Quality Threshold (QT)* algorithm [6] for clustering. This algorithm is good for online settings, since it is efficient (with time to decide which cluster to add a document to linear in the number of clusters) and dynamic (allowing the creation of new clusters over time). It has a thresholding parameter that can be tuned for performance. We use a batch-style implementation of this algorithm *before* beginning to read the stream in order to initialize clusters with the tweets we retrieved as described in Section 3.1.2.

3.2.1 Batch-style implementation of QT algorithm

Algorithm 1 demonstrates the clustering process for this phase. The first cluster is built with the first tweet in the collection. The algorithm iterates over all available tweets (recall that these are the ones retrieved from the Twitter API for the profile in Section 3.1.2), and each one that is "close enough" (as measured by cosine similarity using only term frequencies) to be within the specified diameter (given by a threshold on similarity) is added to that cluster and removed from the collection. As long as there are more tweets in the collection, new clusters are created. The process repeats until no tweets remain in the collection.

Using complete linkage distance (from the tweet and the furthest tweet in a cluster) ensures that the clusters are tightly bound and only tweets with high similarity exist in the same cluster. Thus even a small fragment of new information will help the tweet in getting tagged as novel. This keeps recall high, allowing more tweets to be appended to enrich the profile.

Once we have the clusters for a profile, we pick the earliest tweet by timestamp from each cluster to be the cluster representative. The text from these representatives are then concatenated to form a single text document, and the terms in this document are stored with a profile in the "Info" field in our Lucene index (Table 1).

```
Data: Tweet(s) retrieved from Twitter Search API for
      a profile, scoreThreshold
Result: Cluster(s) of Tweets
collection ← Twitter.search(profile.title)
clusters = {};
$i$ = -1;
while !collection.isEmpty() do
    $i$++;
    $C_i$ = new Cluster();
    forall the newTweet: collection do
        check = true;
        forall the tweet: $C_i$.tweets do
            score = CosineSimilarityScore(tweet.text,
            newTweet.text);
            if score < scoreThreshold then
                check = false;
                break;
            end
        end
        if check then
            $C_i$.tweets.add(newTweet);
            collection.remove(newTweet);
        end
    end
    clusters.add($C_i$);
end
return clusters;
```

Algorithm 1: Batch-style implementation of QT algorithm.

3.3 Retrieving profiles

After the steps described above, our profile index is ready to be processed against the live stream of tweets. We query our index using each incoming tweet and retrieve the user profile(s) that best match that tweet, if any. Several different tf-idf scores are computed and assembled into one final score (Section 3.3.1). Tweets above a particular score threshold are assumed to fit the respective profile and are then reviewed for novelty (Section 3.3.2) before being pushed to the user (Section 3.3.3).

3.3.1 Scoring profiles

As described above, profiles are indexed with multiple fields. While looking for the best-match profile(s), we compute three different tf-idf scores from these fields:

- Multi field score (`mfScore`): Lucene scoring using the Doc + Info fields.
- Title score (`tScore`): Lucene scoring using the Title field.
- Pseudo phrase score (`adjustedScore`): Adjusted version of the title score.

The pseudo phrase score is an adjusted version of the title score which emphasizes more term matches. It is similar to a pseudo phrase search, where instead of looking for the entire phrase, we look for a fragment of the phrase greater than a particular size. We do this in two steps. First, we use the `minShouldMatchQuery` function of Lucene to look for profile titles that have at least x terms in common with the new tweet. This x is a float value between 0.35 and 1 and is inversely proportional to the tweet length. It is computed as follows:

$$\min(1/\texttt{newTweet.length} + 0.35, 1)$$

We then multiply the score returned by Lucene with the ratio of new tweet length to the title length. It is computed as follows:

$$\min(\texttt{newTweet.length}/\texttt{title.length}, 1.0)$$

The second step is meant to take care of cases in which the tweet is short and matches entirely with some topic title containing those words. For example, the tweet is ‘same’ and the title is ‘same sex marriage’—such a match should be discounted.

These scores are then assembled into a final score, which will be used for evaluating the tweet’s appropriateness for the profile. The final score is computed in one of the following ways:

1. `finalScore = mfScore*2+tScore`
2. `finalScore = mfScore*2+adjustedScore`

For a tweet to be matched to a profile, these scores should exceed some threshold. In addition, they must be checked for novelty against tweets that have already been seen.

3.3.2 Novelty detection

Tagging a tweet novel during run-time requires more stringent measures. Every profile maintains clusters of tweets that have been previously found suitable for it. An incoming tweet is only tagged as a novel tweet if it does not seem to match any cluster in that profile.

To ensure high precision, we do the exact opposite of the complete linkage we used previously. Instead, the distance between a tweet and the cluster is the distance from the tweet and the *closest* tweet in that cluster. Once again, the distance between two tweets is the cosine similarity score computed using only term frequencies. Thus enough term overlap with any existing cluster will tag the tweet as redundant. Algorithm 2 shows pseudocode; lines 7–18 check whether a tweet matches existing clusters, and line 21 marks a tweet as novel if it does not match existing clusters.

```
   Data: newTweet, profile, scoreThreshold
   Result: Identify if a new Tweet is novel for the given
           profile
 1 clusters ← profile.clusters;
 2 clusterScores = {};
 3 if clusters.isEmpty() then
 4     clusters.add(new Cluster(newTweet));
 5     novelTweet(newTweet, profile);
 6 else
 7     forall the cluster: clusters do
 8         clusterScore = 0;
 9         forall the tweet: cluster.tweets do
10             score = CosineSimilarityScore(tweet.text,
               newTweet.text);
11             if score > clusterScore then
12                 clusterScore = score;
13             end
14         end
15         if clusterScore ≥ scoreThreshold then
16             clusterScores.put(clusterScore,cluster);
17         end
18     end
19     if clusterScores.isEmpty() then
20         clusters.add(new Cluster(newTweet));
21         novelTweet(newTweet, profile);
22     else
23         bestMatch = clusterScores.highestKey();
24         closestCluster = clusterScores.get(bestMatch);
25         closestCluster.tweets.add(newTweet);
26     end
27 end
```

Algorithm 2: Identification of novel tweets and dynamic cluster updating.

Push notifications have a low saturation rate and thus we only wish to issue notifications for tweets that are completely novel. Therefore the amount of overlap necessary for a tweet to be considered redundant is difficult to predict without training data. This is a parameter that can be tuned to maximize effectiveness.

3.3.3 Selecting tweets for push notifications and daily digests

A push notification is issued as soon as it is identified whereas daily digests are computed at the end of the day. An incoming tweet found both suitable (by the `finalScore` described in Section 3.3.1) and novel (by the cluster similarity thresholding described in Section 3.3.2) for a profile is pushed to the user instantly, as long as the daily quota has not been exhausted. (Retweets are ignored for push notifications.) Push notifications have a low saturation rate and thus we set high score thresholds and use stringent policies while detecting novelty. Again, it is difficult to predict these thresholds without any training data and thus our offline experiments train on historical data to tune these parameters.

Digests are created daily at the end of the day. For every profile, all the new clusters (those that have not been used to create digests for previous days) are read at the end of the day, and the best tweet, i.e. the one with the highest final score, is selected from each cluster to be added to the digest. Once a tweet is picked from a cluster, the cluster is marked as used and that cluster will never be read again for creating digests. These selected tweets are then sorted on either the arrival time or the final score. This differed from experiment to experiment. The top 100 tweets are selected, if any, for each digest. For both scenarios, there is a possibility that no tweets are published for some profiles on some days. This is the ideal behavior for a recommender system—it is expected to stay quiet on days when no relevant or novel tweets are found.

3.4 Dynamic index updates

Our system is designed to adapt over time to accommodate new information and capture changes in social trends as soon as possible. Our Lucene index is frequently updated during run-time to reflect any new information from identified matching tweets. Once a tweet is found suitable for a profile, its text is appended to the term vector. Irrespective of its novel or retweet status, its hashtag contents are used to update hashtag counts and the "Title" field in the index.

3.4.1 Updating clusters

Profile clusters are kept up-to-date with the stream as well. In addition to identifying novel tweets, Algorithm 2 describes how profile clusters are updated given tweets. In particular, a tweet that is identified as novel results in the creation of a new cluster initialized with that tweet (line 20). A tweet that is not identified as novel but that has sufficient similarity to at least one existing cluster is added to the cluster it matches most closely (lines 23–25). In this way, new aspects or facets that emerge over time are allowed to form their own clusters, while clusters that already exist can experience "drift" in their term content according to the most recent tweets.

4. EVALUATION SETUP

We conduct experiments on a 10-day-long live stream from July 20, 2015, 00:00:00 UTC to July 29, 2015, 23:59:59 UTC, consisting of approximately 14 million tweets sampled from the full Twitter stream during the same period. This is the data used for the TREC 2015 Microblog track. 225 standard TREC-style topics were generated by TREC organizers for the track. These topics were formulated around a user's description of information on which he would like to receive timely updates. Each topic consists of three fields:

1. Title: a keyword query highlighting the information need.
2. Description: a sentence summarizing the information need.
3. Narrative: a paragraph-length explanation of area that the user is interested in tracking, including what sort of information should be considered relevant.

As mentioned in Section 3, we used the title and description fields to index topics.

Our experiments were conducted in two separate settings: online and offline. Online experiments performed recommendation on the live public sample Twitter stream for the 10 designated days of Microblog track, pushing tweets instantaneously as they were identified as relevant and novel. We saved a crawl of the same 10-day-long stream in order to replay it for offline experimentation.

In both settings, recommendations were generated for two scenarios: A) instant push notifications (up to 10 per day per user) and B) personalized daily digests with no more than 100 items at the end of the day, summarizing updates for the user's area(s) of interest. Scenario A models the task of receiving real-time notifications on handheld devices such as smartphones or inserting relevant tweets into a user's feed. Scenario B on the other hand could be considered as a tweet timeline generation task where the activity log over a period of time is summarized for the user.

4.1 Online evaluation

We ran three experiments in the online setting. These three experiments were conducted by running three parallel instances of our system on different machines. A pilot study by the track organizers verified that different instances of Twitter's public sample stream received the same tweets[21]. These experiments were one of the first ones to be done at this scale in an online setting.

Our three online experiments are referred as Run1, Run2, and Run3 for the rest of the paper. A run can be considered as a synonym for an experiment. As each run performs recommendation in two scenarios, all runs with the suffix 'A' will indicate the experiments output for scenario A. Similarly, all the runs with suffix 'B' will indicate the experiments output for scenario B. We might also generalize Run1A and Run1B as Run1 for simplicity as they are essentially generated by the same instance and have the same properties; they differ only in the score threshold used when deciding whether to emit a tweet.

These experiments evaluate the impact of differences in data used for profile expansion and different scoring techniques, while holding threshold parameters constant. Table 2 summarizes the score thresholds and the data used for profile expansion for these runs. Run1 and Run2 use documents extracted using the CustomSearch API of Google during profile expansion, whereas Run3 uses documents extracted from ClueWeb12 dataset using Indri indexes. In all three runs, a tweet is considered suitable for a profile if its `mfScore` is 0.8 or higher. Once it is identified as a good match, its contents are used for updating the profile; it is then further assessed for novelty before selecting it for push notification or daily digest or both. Run2 uses the adjusted title score `adjustedScore` to compute the final score (version 2 in Section 3.3.1) whereas Run1 and Run3 use the unadjusted score (version 1). Finally, a tweet qualifies for a push notification if (a) it is not a retweet, (b) has a final score or 2.2 or higher, and (c) has similarity lower than 0.65 with any existing cluster in the profile. Thus the requirements for push notification are quite stringent.

Scores and clustering thresholds were tuned manually due to the lack of training data before the online evaluation phase. They were tuned by eye based on the systems performance for the given sample profiles and a few dummy profiles. These dummy profiles were created by us based on trending stories at that particular instant. These sample

Runtag	Scenario A threshold	Scenario B threshold	Clustering threshold	Final score function	Document source	Term vector & hashtags
Run1	2.2	0.8 (mf)	0.65	mfScore*2+tScore	Google	Twitter
Run2	2.2	0.8 (mf)	0.65	mfScore*2+adjustedScore	Google	Twitter
Run3	2.2	0.8 (mf)	0.65	mfScore*2+tScore	ClueWeb12	Twitter

Table 2: Online run parameters and data used for profile expansion.

and dummy profiles were expanded and indexed using the same techniques.

4.2 Relevance judgements and metrics

The task required the assessors to model topics around information needs that were likely to produce tweets during the online evaluation period. Predicting popularity for these forthcoming needs or for that matter any need in the future is a tough task. Thus more topics were generated than required, with the intention of only using those that received a good response. Of the 225 generated topics, 50 topics were selected for evaluating the participating systems. These 50 included several topics with zero or only a few relevant tweets. They were retained with the purpose of evaluating the systems ability to handle such cases.

Relevance judgements were generated by traditional pooling methodology, with a single pool for both scenario runs. As it was verified that every handle of the public stream received the same tweets, participating systems could run experiments independently and submit the results to NIST at the end of the evaluation period. This stream was also crawled by the organizers to generate the official collection for evaluation purposes. Up to 10 tweets per day from scenario A runs contributed to the judgement pool. Tweets from Scenario B runs were collected in a round-robin fashion, where the 1st-ranked tweet from every run for day 1 was selected, then rank 2 and so on. Tweets were collected across days until 85 tweets were collected for every profile.

English tweets were judged by assessors as "non-relevant", "relevant", or "highly-relevant"; all unjudged tweets were considered as "non-relevant". These judged tweets were then manually clustered using the semantic clustering protocol for the tweet timeline generation task from the TREC 2014 Microblog track [15]. Within these clusters, the earliest tweet is considered to be novel, while all other tweets are considered to be redundant with respect to the previous tweets. Based on this definition, evaluation metrics were designed to penalized tweets for redundancy in both scenarios. For both the scenarios, the score for a run is computed as the average of the score across all the topics and score for a topic is computed as the average of the score across all days.

Scenario A runs are evaluated using two metrics. Expected latency-discounted gain (ELG) from the TREC temporal summarization track [5] is the primary metric. This is analogous to a traditional precision score, but allowing for graded judgments and penalizing latency. It is computed as the sum of the relevance gains over the total number of tweets returned, multiplied by a linear latency penalty. Note that only one tweet from every manually-generated cluster is eligible for the gain. The metric is operationalized as follows:

$$ELG = \frac{1}{N}\sum_{i=1}^{N} G(t_i) \cdot \max(0, (100 - d)/100)$$

Here N is the number of tweets returned. $G(t_i)$ is the gain for tweet t_i based on its relevance: 0 for non-relevant, 0.5 for relevant, and 1.0 for highly-relevant. d is the delay computed in minutes as the difference between the time a tweet was created and the time it was pushed to the user. This penalty decays linearly so that no gain is received for a tweet pushed 100 minutes after its creation.

Normalized cumulative gain (nCG) is the second evaluation metric for scenario A. It is the sum of the gains divided by the total possible gain, where the gain is computed as above. Normalized discounted cumulative gain (nDCG), a more traditional IR metric, is used for evaluating scenario B runs. For every topic, nDCG@10 scores are averaged across all days and then the average of these scores is considered as the score for the run.

For both scenarios, there is a possibility that no tweets are published by a system for some profiles on some days. This is the ideal behavior for a recommender system and if no relevant tweets appear in the judgement pool for the same, the system should be rewarded for its behavior. Here are a few corner cases for handling these situations in evaluation:

1. If there are relevant tweets for a particular day
   - and the system returns 1 or more tweets: score as above.
   - and the system does not return any tweet: it gets a score of 0.
2. If there are `NO` relevant tweets for a particular day
   - and the system returns tweets: score as above.
   - and the system does not return any tweet: it gets a perfect score of 1.

4.3 Offline evaluation

We also conducted offline experiments on a crawl of the same 10-day-long Twitter public sample stream. Tweets received during the evaluation period were written to disk and compressed on hourly basis. During offline experimentation, these files were read sequentially and tweets from these files were replayed just like the live stream. The same sets of tweets and documents were used for profile expansion that were retrieved before the online evaluation period.

Our first set of experiments were done simply to test whether we could replicate our online runs in this offline setting. Since we can no longer use tweet creation time to measure latency, instead the time delay is computed as the difference between the time a tweet was read from disk and the time it was selected for push notification. The sum of this delay and the tweet creation time was set as the notification time. This difference might affect the results for scenario A—it is possible that I/O issues during the online experiment could have created unexpected delays that are not present in the offline setting—but it should not have any impact on scenario B. However, it turns out that we could replicate our online runs almost identically. Online and offline runs with the same parameter settings had very similar results with small but not significant differences for both scenarios. We speak more about these results in Section 5.

The advantage of offline experiments with TREC data is that it allows us to have training and testing phases to better tune our system parameters. We split the relevance judgements historically by day into training and testing parts and modified the evaluation files to do the same. For some experiments we trained on the first 5 days of the stream and tested on the last 5 days; for others we trained on just the first 2 days and tested on the last 8.

Parameters we trained were the clustering threshold used to determine novelty in Algorithm 2 (ranging from 0.5 to 0.9) and the score thresholds for scenarios A and B (ranging from 2.2 to 5) to determine whether a tweet was suitable for a profile.

Additionally, we had two different testing phases:

1. "fresh start testing", in which the testing phase uses the best parameters from the training phase, but clears out the profiles of any data that was added to them during training;

2. "continued testing", in which the testing phase continues to use data that was added to profiles during training.

The former models the case in which a system is trained using some information needs in order to be useful for new information needs. The latter models the case in which a system is trained on information needs that will continue to be processed in the future.

5. RESULTS AND ANALYSIS

Our system worked effortlessly under both online and offline settings. The majority of push notifications were issued without any delay. Delays associated with push notifications, if any, were insignificant. Our system not only performed competitively in the online setting with no training; it also achieves more than 20% improvement when trained on historical data.

5.1 Online runs

Table 3 presents our results in comparison with the median performance of all TREC submissions in the online setting. Run1 and Run2, which use Google documents to expand the profiles, both performed better than the TREC median by ELG, though worse by nCG. Run2 has higher ELG on average compared to Run1 and Run3, and was also the 3rd best automatic run among all those submitted for scenario A [16]. Run2's better performance implies that our pseudo phrase score works well.

5.2 Offline runs

Our first set of offline experiments were done to see if we could replicate online runs under offline settings. Since Run2 was our best performing run under the online setting, we chose to experiment more with the same set of expanded profiles and scoring techniques (pseudo phrase score) used for this run. Table 4 shows the average performance of Run2 under both online and offline settings. They have very similar results; while there are differences both topic-wise and on average for both the scenarios, the differences are not significant. We begin by analyzing these differences.

	scenario A		scenario B
Run	ELG	nCG	nDCG@10
Run1	0.2505	0.2070	0.1966
Run2	0.2670	0.2064	0.2026
Run3	0.2259	0.1910	0.1778
TREC Median	0.2314	0.2170	0.1781

Table 3: ELG and nCG scores for scenario A, and nDCG@10 for scenario B, for our online runs compared to the median performance of all TREC submissions.

Run	ELG	nCG	nDCG@10
Run2 online	0.2670	0.2064	0.2026
Run2 offline	0.2662	0.2108	0.1953

Table 4: Replication of online Run2 under offline settings.

We first analyze the effect of simulating a stream from static data on disk. The main issue is the possibility of time delays in the stream that cannot be simulated, which may result in offline runs being scored higher due to lower latency. To analyze the effect of this, Table 5 presents a summary of the time delays that occurred under both online and offline settings. The first two rows represent two of our online runs (Run1 and Run3). With a median delay of 13 and 37 seconds, 97% of the notifications were issued within a minute and thus with zero latency penalty under the online setting.

Offline runs (rows 3 and 4 in Table 5) replayed the 10-day-long stream over the course of approximately 35 hours, thus processing more tweets per second compared to the online setting. These runs had the same (like online) parameters but the time delays observed under offline setting were even less than the online setting, as the offline runs face a more steady stream as files were read sequentially opening a new file only when the previous one was processed. In contrast, online runs probably faced bursts of tweets during peak hours causing the remaining 3% of delays.

The negligible time delays confirm that almost all of our notifications are pushed instantaneously in both the settings, and support the validity of performing experiments offline. Moreover, despite the fact that time delays in the offline setting are practically non-existent (which may affect the role latency plays in the evaluation), it is apparent from the results in Table 4 that the difference does not significantly affect results.

The difference in time delays does not explain all differences in the result. It is also the case that the dynamic nature of our system makes it prone to problems such as profile drift. We performed experiments with different starting points in time and found that results could be very different (not shown). On manually evaluating this difference, we found differences in the cluster structure and the leading terms in them; furthermore, these differences increased over the days. This indicated that for low score thresholds, even a few false positive tweets can change the profile substantially, and these changes then attract different tweets now suitable to the updated profile but actually irrelevant to the

Setting	Min	Mean	Median	97% Qu.	99% Qu.	Max
Online	0.00	0.00	10.49	13.00	120.29	1708.00
Online	0.00	0.00	15.81	37.00	259.35	2150.00
Offline	0.00	0.00	0.005	0.00	0.00	1.00
Offline	0.00	0.00	0.004	0.00	0.00	1.00
Offline	0.00	0.00	0.007	0.00	0.00	14.00

Table 5: Min, mean, median, 97% quartile, 99% quartile and max time delays (in seconds) for two online and 3 offline runs.

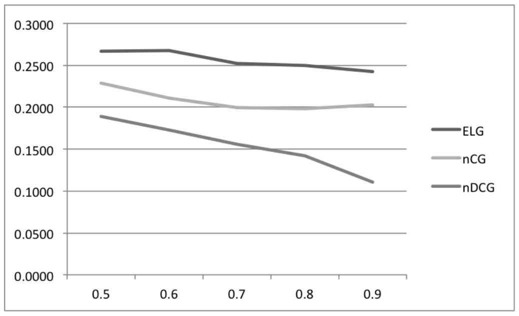

Figure 1: Effect of different clustering thresholds on systems performance.

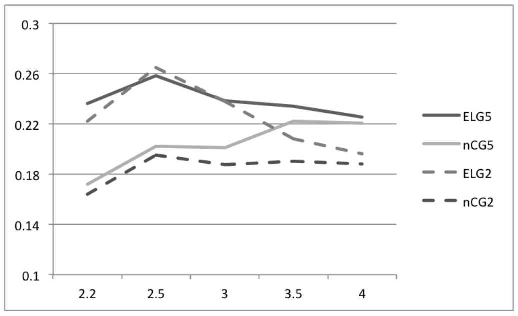

Figure 2: ELG and nCG scores for scenario A when trained with different score thresholds on the first 5 and 2 days of the stream.

information need this profile was modeled around.

This indicates that even minor differences in the stream such as a handful of extra or fewer tweets will affect the way clusters are formed, thus making it impossible to replicate runs with 100% similar performances. This is especially true when the thresholds are low.

5.2.1 Clustering threshold

Clustering plays an important role, and thus our first experiment was to determine the best clustering threshold before tuning other parameters. We performed experiments with different clustering thresholds ranging from 0.5 to 0.9. Figure 1 summarizes the effects of these thresholds on the system's performance. We found that 0.5 was the best clustering threshold. Along with the decline in performance with higher thresholds, the run-time of our experiments increased significantly with higher thresholds: the higher the threshold, the more clusters were created, which directly affected the time required for novelty detection. The experiment with a 0.9 threshold took over 7 days to replay the stream (compared to the 35 hours for the 0.65 threshold in our online experiments). This behavior can be observed from the delay summary mentioned in Table 5 as well. Runs in rows 3, 4 and 5 had 0.65, 0.5 and 0.9 values respectively for clustering thresholds, and the max time delay for the 0.9 threshold is clearly higher (though still very low). Since the clustering threshold of 0.5 seems clearly the best, all the rest of our results use that as the clustering threshold.

5.2.2 Score threshold

Recall that while looking for the best-match profile(s), we compute three different tf-idf scores and assemble them into one final score. This final score is used for determining the suitability of a tweet to a profile. For the online setting, a tweet was considered suitable for a profile if its mfScore was 0.8 or higher and qualified for push notification if it was not a retweet and had a final score or 2.2 or higher. These values were set without any training data. To find out what thresholds fit the best, we experimented with different final score thresholds on our training splits of 2 and 5 days respectively.

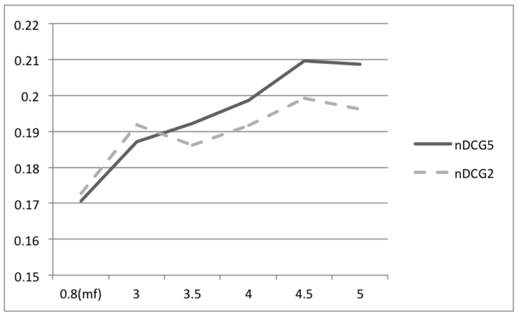

Figure 3: nDCG@10 scores for scenario B when trained with different score thresholds on first 5 and 2 days of the stream.

Figure 2 plots systems performance across scenario A metrics for thresholds ranging between 2.2 and 4. Similarly Figure 3 plots systems performance for scenario B for thresholds ranging from 0.8 (mf) to 5. Here 0.8 is not a final score threshold. It is an mfScore threshold and as mentioned before, all the tweets above this score, irrespective of the final score were considered suitable for a profile and thus eligible for daily digests. In contrast, we now set final score thresholds for scenario B as well.

Based on training on the first 5 days, we found a final score threshold of 3.5 to work best for scenario A. With a subtle drop in ELG, we saw major improvements in nCG. Similarly performance for scenario B increased with higher thresholds and saturated around 4.5. Training on the first two days revealed similar results for scenario B but not for

A. As seen in Figure 2, ELG peaks at 2.5 and drops after that when trained on first 2 days, whereas nCG is almost the same after 2.5.

We then apply these parameters on testing splits to see if these improvements translate to the testing data as well. Tables 7 and 6 present top results based on the training thresholds. We set our system's performance with the original Run2 parameters on the testing splits as the testing baselines. As mentioned above, testing was done in two ways: 1) fresh start or 2) continued testing. In fresh start testing, the system starts (on day 3 or day 6 depending on the number of days used for training) with the updated parameters but without any data from the training days; it starts with the expanded profiles from day 0 and perform recommendation for these profiles. During continued testing, the system continues recommendation on the existing profiles (with all the data from the training days, i.e the updated index and clusters of tweets found suitable on training days).

As seen in Tables 7 and 6, lack of information in profiles has a considerable impact on the baselines (rows 1 & 4), but it does not affect system performance with higher thresholds. This indicates that the system is more robust with higher thresholds. Major improvements can be seen in scenario B results. Significant improvements can also be seen on nCG metric. Improvements as high as 50% were scene for scenario B. In scenario A, an 18% increase in nCG came with a drop of 2% in ELG.

5.2.3 Comparison to TREC Runs

Finally, table 8 compares our system with the best performing runs at TREC 2015 evaluated on only the last five days of the stream[1]. With training, our performance jumped 13 ranks (from 0.2188 nDCG to 0.2638) to be the 3rd best run in scenario B. Training does not lead to as much success in terms of ranking for scenario A; since the increase in nCG came with a drop in ELG.

6. CONCLUSION AND FUTURE WORK

We have introduced a novel framework for tracking Twitter's live stream to generate real-time recommendations. With over 500 million tweets posted per day, Twitter is a rich and a rapid source of mostly-textual data. Analyzing data at this scale requires automatic and intelligent systems that can track these data streams. We show that filtering of relevant information even at this scale can be done almost instantaneously. We also present effective clustering techniques for handling redundancy, detecting novelty, and dynamically updating tracked profiles during run-time. Offline experiments in addition to our online experiments proved that our system not only performs competitively in online setting but also has significant improvement as high as 50% when trained on historical data.

With the use of simple techniques our system does quite well, but there is much more to achieve here. Real-time TDT and recommendation techniques have a lot of room for improvement and our work presents a potential starting point for solving active research problems in these areas. Robustness is one of them. Our future work primarily includes experimenting with better scoring techniques not only to improve filtering accuracy but also to avoid problems such as concept drifts. We could make use of clusters structure and properties to evaluate importance of each cluster and the aspect of information need it represents. Cluster depths can act as signals for cluster significance and can be used for ranking cluster representatives.

Recommender systems have an unending process of learning over time to improve classification accuracy. We plan to experiment with automatic parameter tuning at frequent time intervals using the already classified information. In addition, social streams have many more signals such as hashtags, screennames, and user mentions. We could add screennames and user mention to our index the way we add hastags and use these signals to support text relevance. We could also experiment with tweet signals such as likes and retweets. We could watch identified tweets for these signals to validate our selection and then push it to the user. Not only push notifications but daily digests can also benefit from these signals. Digests are computed after longer time intervals allowing these signals to mature and thus help better in validating tweet's usefulness.

As stated in the introduction, the volume of information generated is so high that it is inevitable that one will miss things that would be interesting to them. Our work is a big step towards making this possible. We aim towards making this a comprehensive microblog recommendation system in the near future.

[1]We used the submitted run files acquired from TREC organizers to evaluate them on the last five days of the stream.

Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant IIS-1350799. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

7. REFERENCES

[1] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *Knowledge and Data Engineering, IEEE Transactions on*, 17(6):734–749, 2005.

[2] J. Allan. *Topic detection and tracking: event-based information organization*, volume 12. Springer Science & Business Media, 2012.

[3] J. Allan, J. G. Carbonell, G. Doddington, J. Yamron, and Y. Yang. Topic detection and tracking pilot study final report. 1998.

[4] L. AlSumait, D. Barbará, and C. Domeniconi. On-line lda: Adaptive topic models for mining text streams with applications to topic detection and tracking. In *Data Mining, 2008. ICDM'08.*, 2008.

[5] J. Aslam, F. Diaz, M. Ekstrand-Abueg, R. McCreadie, V. Pavlu, and T. Sakai. Trec 2014 temporal summarization track overview. In *Proc. TREC*, 2014.

[6] L. Bednarik and L. Kovacs. Efficiency analysis of quality threshold clustering algorithms. *Production Systems and Information Engineering, University of Miskolc*, 6:1785–1270, 2012.

[7] K. Chen, T. Chen, G. Zheng, O. Jin, E. Yao, and Y. Yu. Collaborative personalized tweet recommendation. In *Proc. SIGIR*, 2012.

[8] C. Cieri, S. Strassel, D. Graff, N. Martey, K. Rennert, and M. Liberman. Corpora for topic detection and tracking. In *Topic detection and tracking*, pages 33–66. Springer, 2002.

[9] E. Elsawy, M. Mokhtar, and W. Magdy. Tweetmogaz v2: Identifying news stories in social media. In *Proc. CIKM*, 2014.

Scenario A threshold	Scenario B threshold	Clustering threshold	Profile start	ELG		nCG		nDCG@10	
2.2	0.8(mf)	0.65	continued	0.2737		0.2173		0.1911	
3.5	4.5	0.5	continued	0.2684	-1.94%	0.2580	18.73%*	0.2387	24.91%
3.5	5.0	0.5	continued	0.2684	-1.94%	0.2580	18.73%*	0.2436	27.47%
2.2	0.8(mf)	0.65	fresh	0.2735		0.2194		0.1595	
3.5	4.5	0.5	fresh	0.2574	-5.89%	0.2481	13.08%*	0.2383	49.40%*
3.5	5.0	0.5	fresh	0.2574	-5.89%	0.2481	13.08%*	0.2407	50.91%*

Table 6: Testing results with and without training data on the last 8 days of the stream. Rows 1 and 4 represent the baseline: the performance of the original online Run2 parameters limited to the last 8 days of the stream. A * indicates statistically significant improvement over the baseline by a paired t-test at the 0.05 level.

Scenario A threshold	Scenario B threshold	Clustering threshold	Profile start	ELG		nCG		nDCG@10	
2.2	0.8(mf)	0.65	continued	0.2860		0.2388		0.2044	
3.5	4.5	0.5	continued	0.2765	-3.32%	0.2658	11.31%	0.2599	27.15%
3.5	5.0	0.5	continued	0.2765	-3.32%	0.2658	11.31%	0.2638	29.06%
2.2	0.8(mf)	0.65	fresh	0.2948		0.2368		0.1899	
3.5	4.5	0.5	fresh	0.2765	-6.21%	0.2603	9.92%	0.2553	34.44%
3.5	5.0	0.5	fresh	0.2765	-6.21%	0.2603	9.92%	0.2592	36.49%*

Table 7: Testing results with and without training data on last 5 days of the stream. The first line of each group is the baseline: the performance of the original online Run2 parameters limited to the last 5 days of the stream. A * indicates statistically significant improvement over the baseline by a paired t-test at the 0.05 level.

	Scenario A			Scenario B
Run	ELG	nCG	Run	nDCG@10
SNACSA	0.3261	0.3559	SNACS_LB	0.3864
UWaterlooATDK	0.326	0.2749	SNACS	0.3375
...			Run2	0.2638
Run2	0.2765	0.2658	umd_hcil_run3	0.2627

Table 8: Our performance in comparison with the best performing TREC runs limited to the last 5 days of the stream. Run2 shows our offline performance with the best learned parameters.

[10] U. Hanani, B. Shapira, and P. Shoval. Information filtering: Overview of issues, research and systems. *User Modeling and User-Adapted Interaction*, 11(3):203–259, 2001.

[11] M. Jiang, P. Cui, F. Wang, Q. Yang, W. Zhu, and S. Yang. Social recommendation across multiple relational domains. In *Proc. CIKM*, 2012.

[12] X. N. Lam, T. Vu, T. D. Le, and A. D. Duong. Addressing cold-start problem in recommendation systems. In *Proc. ICUIMC*, 2008.

[13] V. Lavrenko, J. Allan, E. DeGuzman, D. LaFlamme, V. Pollard, and S. Thomas. Relevance models for topic detection and tracking. In *Proc. HLT*, 2002.

[14] T. Leek, R. Schwartz, and S. Sista. Probabilistic approaches to topic detection and tracking. In *Topic detection and tracking*, pages 67–83. Springer, 2002.

[15] J. Lin, M. Efron, Y. Wang, and G. Sherman. Overview of the trec-2014 microblog track. In *Proc. TREC*, 2014.

[16] J. Lin, M. Efron, Y. Wang, G. Sherman, and E. Voorhees. Overview of the trec-2015 microblog track. In *Proc. TREC*, 2015.

[17] W. Magdy. Tweetmogaz: a news portal of tweets. In *Proc. SIGIR*, 2013.

[18] W. Magdy and T. Elsayed. Adaptive method for following dynamic topics on twitter. In *Proce. ICWSM*, 2014.

[19] J. Makkonen, H. Ahonen-Myka, and M. Salmenkivi. Simple semantics in topic detection and tracking. *Information retrieval*, 7(3-4):347–368, 2004.

[20] I. Ounis, C. Macdonald, J. Lin, and I. Soboroff. Overview of the trec-2011 microblog track. In *Proc. TREC*, 2011.

[21] J. H. Paik and J. Lin. Do multiple listeners to the public twitter sample stream receive the same tweets? In *Proc. SIGIR Workshop on Temporal, Social and Spatially-Aware Information Access*, 2015.

[22] J. B. Schafer, D. Frankowski, J. Herlocker, and S. Sen. Collaborative filtering recommender systems. In *The adaptive web*, pages 291–324. Springer, 2007.

[23] I. Soboroff, I. Ounis, J. Lin, and I. Soboroff. Overview of the trec-2012 microblog track. In *Proc. TREC*, 2012.

[24] I. Uysal and W. B. Croft. User oriented tweet ranking: a filtering approach to microblogs. In *Proc. CIKM*, 2011.

[25] X. Wang, L. Tokarchuk, F. Cuadrado, and S. Poslad. Exploiting hashtags for adaptive microblog crawling. In *Proc. ASONAM*, 2013.

[26] C. L. Wayne. Multilingual topic detection and tracking: Successful research enabled by corpora and evaluation. In *LREC*, 2000.

[27] R. Yan, M. Lapata, and X. Li. Tweet recommendation with graph co-ranking. In *Proc. ACL*, 2012.

[28] Y. Zhang, J. Callan, and T. Minka. Novelty and redundancy detection in adaptive filtering. In *Proc. SIGIR*, 2002.

[29] X. Zhou, S. Wu, C. Chen, G. Chen, and S. Ying. Real-time recommendation for microblogs. *Information Sciences*, 279:301–325, 2014.