On Summarization and Timeline Generation for Evolutionary Tweet Streams

Zhenhua Wang, Lidan Shou, Ke Chen, Gang Chen, and Sharad Mehrotra

Abstract—Short-text messages such as tweets are being created and shared at an unprecedented rate. Tweets, in their raw form, while being informative, can also be overwhelming. For both end-users and data analysts, it is a nightmare to plow through millions of tweets which contain enormous amount of noise and redundancy. In this paper, we propose a novel continuous summarization framework called Sumblr to alleviate the problem. In contrast to the traditional document summarization methods which focus on static and small-scale data set, Sumblr is designed to deal with dynamic, fast arriving, and large-scale tweet streams. Our proposed framework consists of three major components. First, we propose an online tweet stream clustering algorithm to cluster tweets and maintain distilled statistics in a data structure called tweet cluster vector (TCV). Second, we develop a TCV-Rank summarization technique for generating online summaries and historical summaries of arbitrary time durations. Third, we design an effective topic evolution detection method, which monitors summary-based/volume-based variations to produce timelines automatically from tweet streams. Our experiments on large-scale real tweets demonstrate the efficiency and effectiveness of our framework.

Index Terms—Tweet stream, continuous summarization, summary, timeline

1 Introduction

Increasing popularity of microblogging services such as Twitter, Weibo, and Tumblr has resulted in the explosion of the amount of short-text messages. Twitter, for instance, which receives over 400 million tweets per day[1] has emerged as an invaluable source of news, blogs, opinions, and more. Tweets, in their raw form, while being informative, can also be overwhelming. For instance, search for a hot topic in Twitter may yield millions of tweets, spanning weeks. Even if filtering is allowed, plowing through so many tweets for important contents would be a nightmare, not to mention the enormous amount of noise and redundancy that one might encounter. To make things worse, new tweets satisfying the filtering criteria may arrive continuously, at an unpredictable rate.

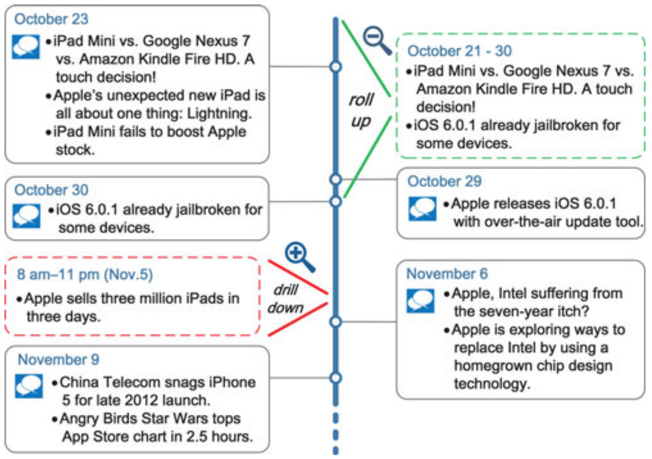
One possible solution to information overload problem is summarization. Summarization represents a set of documents by a summary consisting of several sentences. Intuitively, a good summary should cover the main topics (or subtopics) and have diversity among the sentences to reduce redundancy. Summarization is extensively used in content presentation, specially when users surf the internet with their mobile devices which have much smaller screens than PCs. Traditional document summarization approaches, however, are not as effective in the context of tweets given both the large volume of tweets as well as the fast and continuous nature of their arrival. Tweet summarization, therefore, requires functionalities which significantly differ from traditional summarization. In general, tweet summarization has to take into consideration the temporal feature of the arriving tweets.

Let us illustrate the desired properties of a tweet summarization system using an illustrative example of a usage of such a system. Consider a user interested in a topic-related tweet stream, for example, tweets about "Apple". A tweet summarization system will continuously monitor "Apple" related tweets producing a *real-time timeline* of the tweet stream. As illustrated in Fig. 1, a user may explore tweets based on a timeline (e.g., "Apple" tweets posted between October 22nd, 2012 to November 11th, 2012). Given a timeline range, the summarization system may produce a sequence of timestamped summaries to highlight points where the topic/subtopics evolved in the stream. Such a system will effectively enable the user to learn major news/discussion related to "Apple" without having to read through the entire tweet stream. Given the big picture about topic evolution about "Apple", a user may decide to zoom in to get a more detailed report for a smaller duration (e.g., from 8 am to 11 pm on November 5th). The system may provide a *drill-down* summary of the duration that enables the user to get additional details for that duration. A user, perusing a drill-down summary, may alternatively zoom out to a coarser range (e.g., October 21st to October 30th) to obtain a *roll-up* summary of tweets. To be able to support such drill-down and roll-up operations, the summarization system must support the following two queries: summaries of arbitrary time durations and real-time/range timelines. Such application would not only facilitate easy navigation in topic-relevant tweets, but also support a range of data analysis tasks such as instant reports or historical survey. To this end, in this paper, we propose a new summarization method, *continuous summarization*, for tweet streams.

[1] https://blog.twitter.com/2013/celebrating-twitter7

- *Z. Wang, L. Shou, K. Chen, and G. Chen are with the College of Computer Science and Technology, Zhejiang University, Hangzhou 310027, Zhejiang, China. E-mail: {wzh-cs, should, chenk, cg}@zju.edu.cn.*
- *S. Mehrotra is with the School of Information and Computer Science, University of California at Irvine, Irvine, CA 92697-3425. E-mail: sharad@ics.uci.edu.*

Manuscript received 27 Jan. 2014; revised 19 July 2014; accepted 22 July 2014. Date of publication 4 Aug. 2014; date of current version 27 Mar. 2015.
Recommended for acceptance by P. G. Ipeirotis.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TKDE.2014.2345379



October 23
- iPad Mini vs. Google Nexus 7 vs. Amazon Kindle Fire HD. A touch decision!
- Apple's unexpected new iPad is all about one thing: Lightning.
- iPad Mini fails to boost Apple stock.

October 30
- iOS 6.0.1 already jailbroken for some devices.

8 am–11 pm (Nov.5)
- Apple sells three million iPads in three days.

November 9
- China Telecom snags iPhone 5 for late 2012 launch.
- Angry Birds Star Wars tops App Store chart in 2.5 hours.

October 21 - 30
- iPad Mini vs. Google Nexus 7 vs. Amazon Kindle Fire HD. A touch decision!
- iOS 6.0.1 already jailbroken for some devices.

October 29
- Apple releases iOS 6.0.1 with over-the-air update tool.

November 6
- Apple, Intel suffering from the seven-year itch?
- Apple is exploring ways to replace Intel by using a homegrown chip design technology.

Fig. 1. A timeline example for topic “Apple”.

Implementing continuous tweet stream summarization is however not an easy task, since a large number of tweets are meaningless, irrelevant and noisy in nature, due to the social nature of tweeting. Further, tweets are strongly correlated with their posted time and new tweets tend to arrive at a very fast rate. Consequently, a good solution for continuous summarization has to address the following three issues: (1) *Efficiency*—tweet streams are always very large in scale, hence the summarization algorithm should be highly efficient; (2) *Flexibility*—it should provide tweet summaries of arbitrary time durations. (3) *Topic evolution*—it should automatically detect sub-topic changes and the moments that they happen.

Unfortunately, existing summarization methods cannot satisfy the above three requirements because: (1) They mainly focus on static and small-sized data sets, and hence are not efficient and scalable for large data sets and data streams. (2) To provide summaries of arbitrary durations, they will have to perform iterative/recursive summarization for every possible time duration, which is unacceptable. (3) Their summary results are insensitive to time. Thus it is difficult for them to detect topic evolution .

In this paper, we introduce a novel summarization framework called *Sumblr (continuouS sUMmarization By stream cLusteRing)*. To the best of our knowledge, our work is the first to study continuous tweet stream summarization. The overall framework is depicted in Fig. 2. The framework consists of three main components, namely the *Tweet Stream Clustering* module, the *High-level Summarization* module and the *Timeline Generation* module.

In the tweet stream clustering module, we design an efficient *tweet stream clustering algorithm*, an online algorithm allowing for effective clustering of tweets with only one pass over the data. This algorithm employs two data structures to keep important tweet information in clusters. The first one is a novel compressed structure called the *tweet cluster vector (TCV)*. TCVs are considered as potential sub-topic delegates and maintained dynamically in memory during stream processing. The second structure is the *pyramidal time frame (PTF)* [1], which is used to store and organize cluster snapshots at different moments, thus allowing historical tweet data to be retrieved by any arbitrary time durations.

The high-level summarization module supports generation of two kinds of summaries: online and historical summaries. (1) To generate online summaries, we propose a *TCV-Rank summarization* algorithm by referring to the current clusters maintained in memory. This algorithm first computes centrality scores for tweets kept in TCVs, and selects the top-ranked ones in terms of content coverage and novelty. (2) To compute a historical summary where the user specifies an arbitrary time duration, we first retrieve two *historical cluster snapshots* from the PTF with respect to the two endpoints (the beginning and ending points) of the duration. Then, based on the difference between the two cluster snapshots, the TCV-Rank summarization algorithm is applied to generate summaries.

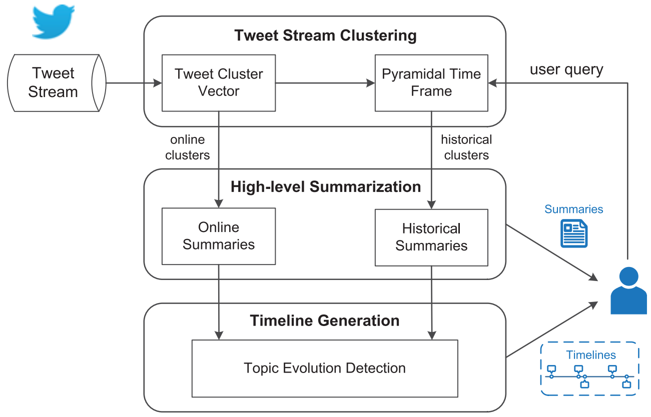


Fig. 2. The framework of Sumblr.

The core of the timeline generation module is a topic evolution detection algorithm, which consumes online/historical summaries to produce real-time/range timelines. The algorithm monitors quantified variation during the course of stream processing. A large variation at a particular moment implies a sub-topic change, leading to the addition of a new node on the timeline. In our design, we consider three different factors respectively in the algorithm. First, we consider variation in the main contents discussed in tweets (in the form of summary). To quantify the *summary-based variation* (SUM), we use the *Jensen-Shannon divergence* (JSD) to measure the distance between two word distributions in two successive summaries. Second, we monitor the *volume-based variation* (VOL) which reflects the significance of sub-topic changes, to discover rapid increases (or "spikes") in the volume of tweets over time. Third, we define the *sum-vol variation* (SV) by combining both effects of summary content and significance, and detect topic evolution whenever there is a burst in the unified variation.

The main contributions of this work are as follows:

- We propose a continuous tweet stream summarization framework, namely Sumblr, to generate summaries and timelines in the context of streams.
- We design a novel data structure called TCV for stream processing, and propose the TCV-Rank algorithm for online and historical summarization.
- We propose a topic evolution detection algorithm which produces timelines by monitoring three kinds of variations.
- Extensive experiments on real Twitter data sets demonstrate the efficiency and effectiveness of our framework.

The remainder of this paper is organized as follows. Section 2 reviews the related work. Section 3 explains concepts including TCV and PTF. Section 4 describes our Sumblr framework in detail. The experimental settings and results are presented in Section 5. In Section 6, we conclude the paper.

2 Related Work

In this section, we review the related work including *stream data clustering*, *document/microblog summarization*, *timeline detection*, and *other microblog mining tasks*.

2.1 Stream Data Clustering

Stream data clustering has been widely studied in the literature. BIRCH [2] clusters the data based on an in-memory structure called CF-tree instead of the original large data set. Bradley et al. [3] proposed a scalable clustering framework which selectively stores important portions of the data, and compresses or discards other portions. CluStream [1] is one of the most classic stream clustering methods. It consists of an online micro-clustering component and an offline macro-clustering component. The pyramidal time frame was also proposed in [1] to recall historical micro-clusters for different time durations.

A variety of services on the Web such as news filtering, text crawling, and topic detecting etc. have posed requirements for text stream clustering. A few algorithms have been proposed to tackle the problem [4], [5], [6], [7]. Most of these techniques adopt partition-based approaches to enable online clustering of stream data. As a consequence, these techniques fail to provide effective analysis on clusters formed over different time durations.

In [8], the authors extended CluStream to generate duration-based clustering results for text and categorical data streams. However, this algorithm relies on an online phase to generate a large number of "micro-clusters" and an offline phase to re-cluster them. In contrast, our tweet stream clustering algorithm is an online procedure without extra offline clustering. And in the context of tweet summarization, we adapt the online clustering phase by incorporating the new structure TCV, and restricting the number of clusters to guarantee efficiency and the quality of TCVs.

2.2 Document/Microblog Summarization

Document summarization can be categorized as extractive and abstractive. The former selects sentences from the documents, while the latter may generate phrases and sentences that do not appear in the original documents. In this paper, we focus on extractive summarization.

Extractive document summarization has received a lot of recent attention. Most of them assign salient scores to sentences of the documents, and select the top-ranked sentences [9], [10], [11]. Some works try to extract summaries without such salient scores. Wang et al. [12] used the symmetric non-negative matrix factorization to cluster sentences and choose sentences in each cluster for summarization. He et al. [13] proposed to summarize documents from the perspective of data reconstruction, and select sentences that can best reconstruct the original documents. In [14], Xu et al. modeled documents (hotel reviews) as multi-attribute uncertain data and optimized a probabilistic coverage problem of the summary.

While document summarization has been studied for years, microblog summarization is still in its infancy. Sharifi et al. proposed the Phrase Reinforcement algorithm to summarize tweet posts using a single tweet [15]. Later, Inouye and Kalita proposed a Hybrid TF-IDF algorithm and a Cluster-based algorithm to generate multiple post summaries [16]. In [17], Harabagiu and Hickl leveraged two relevance models for microblog summarization: an event structure model and a user behavior model. Takamura et al. [18] proposed a microblog summarization method based on the p-median problem, which takes posted time of microblogs into consideration.

Unfortunately, almost all existing document/microblog summarization methods mainly deal with small and static data sets, and rarely pay attention to efficiency and evolution issues.

There have also been studies on summarizing microblogs for some specific types of events, e.g., sports events. Shen et al. [19] proposed to identify the participants of events, and generate summaries based on sub-events detected from each participant. Chakrabarti and Punera [20] introduced a solution by learning the underlying hidden state representation of the event, which needs to learn from previous events (football games) with similar structure. In [21], Kubo et al. summarized events by exploiting "good reporters", depending on event-specific keywords which need to be given in advance. In contrast, we aim to deal with general topic-relevant tweet streams without such prior knowledge. Moreover, their method stores all the tweets in each *segment* and selects a single tweet as the summary, while our method maintains distilled information in TCVs to reduce storage/computation cost, and generates multiple tweet summaries in terms of content coverage and novelty. In addition to online summarization, our method also supports historical summarization by maintaining TCV snapshots.

2.3 Timeline Detection

The demand for analyzing massive contents in social medias fuels the developments in visualization techniques. Timeline is one of these techniques which can make analysis tasks easier and faster. Diakopoulos and Shamma [22] made early efforts in this area, using timelines to explore the 2008 Presidential Debates by Twitter sentiment. Dork et al. [23] presented a timeline-based backchannel for conversations around events.

In [24], Yan et al. proposed the evolutionary timeline summarization (ETS) to compute evolution timelines similar to ours, which consists of a series of time-stamped summaries. However, in [24], the dates of summaries are determined by a pre-defined timestamp set. In contrast, our method discovers the changing dates and generates timelines dynamically during the process of continuous summarization. Moreover, ETS does not focus on efficiency and scalability issues, which are very important in our streaming context.

Several systems detect important moments when rapid increases or "spikes" in status update volume happen. TwitInfo [25] developed an algorithm based on TCP congestion detection, while Nichols et al. [26] employed a slope-based method to find spikes. After that, tweets from each moment are identified, and word clouds or summaries are selected. Different from this two-step approach, our method detects topic evolution and produces summaries/timelines in an online fashion.

2.4 Other Microblog Mining Tasks

The emergence of microblogs has engendered researches on many other mining tasks, including topic modeling [27], storyline generation [28] and event exploration [25]. Most of these researches focus on static data sets instead of data streams.

For twitter stream analysis, Yang et al. [29] studied frequent pattern mining and compression. In [30], Van Durme aimed at discourse participants classification and used gender prediction as the example task, which is also a different problem from ours.

To sum up, in this work, we propose a new problem called continuous tweet summarization. Different from previous studies, we aim to summarize large-scale and evolutionary tweet streams, producing summaries and timelines in an online fashion.

3 Preliminaries

In this section, we first present a data model for tweets, then introduce two important data structures: the *tweet cluster vector* and the *pyramidal time frame*.

3.1 Tweet Representation

Generally, a document is represented as a textual vector, where the value of each dimension is the TF-IDF score of a word. However, tweets are not only textual, but also have temporal nature—a tweet is strongly correlated with its posted time. In addition, the importance of a tweet is affected by the author's social influence. To estimate the user influence, we build a matrix based on social relationships among users, and compute the UserRank as in [31].

As a result, we define a tweet t_i as a tuple: $(\mathbf{tv_i}, ts_i, w_i)$, where $\mathbf{tv_i}$ is the textual vector, ts_i is the posted timestamp and w_i is the UserRank value of the tweet's author.

3.2 Tweet Cluster Vector

During tweet stream clustering, it is necessary to maintain statistics for tweets to facilitate summary generation. In this section, we propose a new data structure called *tweet cluster vector*, which keeps information of tweet cluster.

Definition 1. *For a cluster C containing tweets $t_1, t_2, \ldots, t_n$, its* tweet cluster vector *is defined as a tuple: $TCV(C) = (\mathbf{sum\_v}, \mathbf{wsum\_v}, ts1, ts2, n, ft\_set)$, where*

- $\mathbf{sum\_v} = \sum_{i=1}^{n} \mathbf{tv_i}/||\mathbf{tv_i}||$ *is the sum of normalized textual vectors,*
- $\mathbf{wsum\_v} = \sum_{i=1}^{n} w_i \cdot \mathbf{tv_i}$ *is the sum of weighted textual vectors,*
- $ts1 = \sum_{i=1}^{n} ts_i$ *is the sum of timestamps,*
- $ts2 = \sum_{i=1}^{n} (ts_i)^2$ *is the quadratic sum of timestamps,*
- n *is the number of tweets in the cluster, and*
- ft_set *is a* focus tweet set *of size m, consisting of the closest m tweets to the cluster centroid.*

The form of $\mathbf{sum\_v}$ is used for ease of presentation. In fact, we only store the identifiers and sums of values of the words occurring in the cluster. The same convention is used for $\mathbf{wsum\_v}$. To select tweets into ft_set, we use cosine similarity as the distance metric.

From the definition, we can derive the vector of cluster centroid (denoted as $\mathbf{cv}$)

$$\mathbf{cv} = \left(\sum_{i=1}^{n} w_i \cdot \mathbf{tv_i} \right) \Big/ n = \mathbf{wsum\_v}/n. \tag{1}$$

The definition of TCV is an extension of the cluster feature vector proposed in [2]. Besides information of data points (textual vectors), TCV includes temporal information and representative original tweets. As in [2], our TCV structure can also be updated in an incremental manner when new tweets arrive. We shall discuss details on TCV updating in Section 4.1.2.

3.3 Pyramidal Time Frame

To support summarization over user-defined time durations, it is crucial to store the maintained TCVs at particular moments, which are called *snapshots*. While storing snapshots at every moment is impractical due to huge storage overhead, insufficient snapshots make it hard to recall historical information for different durations. This dilemma leads to the incorporation of the *pyramidal time frame* [1]:

Definition 2. *A* pyramidal time frame *stores snapshots at differing levels of granularity depending on the recency. Snapshots are stored into different* orders *varying from 0 to $\lfloor log_\alpha(T) \rfloor$, where T is the time elapsed since the beginning of the stream and α is an integer $(\alpha > 1)$. A particular order of snapshots defines the level of granularity in time at which the snapshots are maintained. The snapshots of different orders are maintained as follows:*

- *Snapshots of the ith order occur at time intervals of α^i. Specifically, each snapshot of the ith order is taken at a moment in time when the timestamp from the beginning of the stream is exactly divisible by α^i.*
- *At any given moment in time, only the last $\alpha^l + 1 (l \geq 1)$ snapshots of order i are stored.*

According to the definition, PTF has two properties: (1) The maximum order of any snapshot stored at timestamp T since the beginning of the stream is $log_\alpha(T)$; (2) The maximum number of snapshots maintained at T is $(\alpha^l + 1) \cdot log_\alpha(T)$. These properties are crucial for system performance. Taking more snapshots (by using a larger α or l) offers better accuracy of time duration approximation, but meanwhile causes larger storage overhead. Therefore, we need to strike a balance between duration accuracy and storage space. Note that we only maintain the current clusters in main memory, and store all historical snapshots in the PTF on disk.

To clarify how snapshots are stored, we give an example here. Let $\alpha = 3$ and $l = 2$, then there are at most $3^2 + 1 = 10$ snapshots in each order. Suppose the stream starts at timestamp 1 and the current timestamp is 86. The stored snapshots are illustrated in Table 1. Redundancy is removed by storing each snapshot only in its highest possible order.

TABLE 1
Example of PTF with $\alpha = 3$ and $l = 2$

Order	Timestamps of snapshots in the same order
4	81
3	54 27
2	72 63 45 36 18 9
1	84 78 75 69 66 60 57 51 48 42
0	86 85 83 82 80 79 77 76 74 73

Note that for more recent timestamps, the time interval between successive snapshots stored in PTF is smaller (finer granularity). This feature of PTF is consistent with the demand that recent summaries should be of higher quality because people usually care more about recent events.

4 The Sumblr Framework

As shown in Fig. 2, our framework consists of three main modules: the tweet stream clustering module, the high-level summarization module and the timeline generation module. In this section, we shall present each of them in detail.

4.1 Tweet Stream Clustering

The tweet stream clustering module maintains the online statistical data. Given a topic-based tweet stream, it is able to efficiently cluster the tweets and maintain compact cluster information.

4.1.1 Initialization

At the start of the stream, we collect a small number of tweets and use a k-means clustering algorithm to create the initial clusters. The corresponding TCVs are initialized according to Definition 1. Next, the stream clustering process starts to incrementally update the TCVs whenever a new tweet arrives.

4.1.2 Incremental Clustering

Suppose a tweet t arrives at time ts, and there are N active clusters at that time. The key problem is to decide whether to absorb t into one of the current clusters or upgrade t as a new cluster. We first find the cluster whose centroid is the closest to t. Specifically, we get the centroid of each cluster based on Equation (1), compute its cosine similarity to t, and find the cluster C_p with the largest similarity (denoted as $MaxSim(t)$).

Note that although C_p is the closest to t, it does not mean t naturally belongs to C_p. The reason is that t may still be very distant from C_p. In such case, a new cluster should be created . The decision of whether to create a new cluster can be made with the following heuristic.

Heuristic 1. The *minimum bounding similarity* (MBS) measures the average closeness between the centroid and the tweets included in the cluster C_p. MBS is used to decide whether t is close enough to C_p: if $MaxSim(t)$ is smaller than it, then t is upgraded to a new cluster; Otherwise, t is added to C_p.

The MBS is defined as $\beta \cdot \overline{Sim(C_p)}$, where β is a bounding factor ($0 < \beta < 1$) and $\overline{Sim(C_p)}$ is the average cosine similarity between the centroid of C_p and the tweets included in C_p. $\overline{Sim(C_p)}$ can be calculated as follows:

$$\overline{Sim(C_p)} = \frac{1}{n}\sum_{i=1}^{n}\frac{\mathbf{tv_i}\cdot\mathbf{cv}}{||\mathbf{tv_i}||\cdot||\mathbf{cv}||} = \frac{\mathbf{cv}}{n\cdot||\mathbf{cv}||}\sum_{i=1}^{n}\frac{\mathbf{tv_i}}{||\mathbf{tv_i}||}$$
$$= \frac{\mathbf{wsum\_v}/n}{n\cdot||\mathbf{wsum\_v}/n||}\cdot\mathbf{sum\_v} = \frac{\mathbf{wsum\_v}\cdot\mathbf{sum\_v}}{n\cdot||\mathbf{wsum\_v}||},$$

where $\mathbf{tv_i}$ is the textual vector of tweet t_i in C_p and $\mathbf{cv}$ is that of the cluster centroid, as defined in Section 3.1 and Equation (1), respectively. $\mathbf{wsum\_v}$, $\mathbf{sum\_v}$ and n are the information stored in TCV (Definition 1).

When creating a new cluster, it is hard to tell whether it is noise or a truly new sub-topic. Actually, the decision cannot be made until more tweets arrive. We discuss how noises are diminished in Section 4.4.

After applying Heuristic 1 the corresponding TCV needs to be updated. For a newly created cluster, its TCV can be initialized easily. For an existing cluster, its components in TCV can also be easily updated in an incremental manner, except the focus tweet set.

Recall that in Definition 1, ft_set is the set of the closest m tweets to the cluster centroid. However, as new tweets are added to the cluster during stream processing, the cluster centroid would change unpredictably. Since we cannot store all tweets in the cluster, it is rather difficult to maintain exact focus tweets. For this reason, we use a heuristic strategy to choose promising candidates instead of exact focus tweets: for the newly absorbed tweet and those already in ft_set, we compute their cosine distances to *the new centroid*, and select the closest m tweets. The advantage of this strategy is that it gives a higher probability for fresh tweets to get into the focus set, which usually represent new statuses of the topic.

The above updating process is executed upon the arrival of each new tweet. Meanwhile, when the current timestamp is divisible by α^i for any integer i, we store the snapshot of the current TCVs into disk and index it by PTF. Algorithm 1 describes the overview of our incremental clustering procedure.

Algorithm 1. Incremental Tweet Stream Clustering

```
Input: a cluster set $C\_set$
while !stream.end() do
    $Tweet\ t = stream.next()$;
    choose $C_p$ in $C\_set$ whose centroid is the closest to $t$;
    if $MaxSim(t) < MBS$ then
        create a new cluster $C_{new} = \{t\}$;
        $C\_set.add(C_{new})$;
    else
        update $C_p$ with $t$;
    if $TS_{current}\%(\alpha^i) == 0$ then
        store $C\_set$ into PTF;
```

During incremental clustering, assume there are N active clusters, the computational cost of finding the closest cluster for every new tweet is $O(Nd)$, where d is the vocabulary size. In addition, the complexity of computing Heuristic 1 and updating TCV is $O(d)$ and $O(md)$ respectively, where

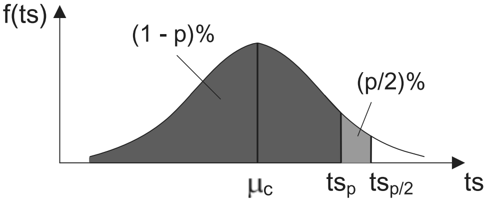


Fig. 3. Probability density function of timestamp.

m is the size of focus set. Then the total cost is $O((N+m)d)$. Because m and d are static, the computational cost depends on N. Similarly, the storage costs in disk (TCV snapshots) and memory (current TCVs) also depend on N.

Given the above analysis, we need to restrict the number of active clusters. We achieve this goal via two operations: *deleting outdated clusters* and *merging similar clusters*. Since the computational complexity of deletion is $O(N)$ and that of merging is $O(N^2)$, we use the former method for periodical examination and use the latter method only when memory limit is reached.

4.1.3 Deleting Outdated Clusters

For most events (such as news, football matches and concerts) in tweet streams, timeliness is important because they usually do not last for a long time. Therefore it is safe to delete the clusters representing these sub-topics when they are rarely discussed. To find out such clusters, an intuitive way is to estimate the average arrival time (denoted as Avg_p) of the last p percent of tweets in a cluster. However, storing p percent of tweets for every cluster will increase memory costs, especially when clusters grow big. Thus, we employ an approximate method to get Avg_p.

Note that the temporal statistics in TCV of a cluster C allow us to compute the mean and standard deviation of timestamps of tweets in C: $\mu_c = ts1/n$, $\sigma_c = \sqrt{ts2/n - (ts1/n)^2}$.

Assuming that the tweet timestamps are normally distributed, we can obtain the arrival time of the qth percentile of the tweets. The *qth percentile* is the value that cuts off the first q percent of the tweet timestamps when they are sorted in ascending order. When $q = 100 - p$, the qth percentile is the start timestamp of the last p percent of tweets (noted as ts_p in Fig. 3). Then, we can approximate Avg_p using the $(100 - p/2)$th percentile (noted as $ts_{p/2}$ in Fig. 3).

Now the problem is transformed into obtaining the value of $ts_{p/2}$. Let $x = ts_{p/2}$ and $p' = (100 - p/2)\%$, we have

$$\begin{aligned} & F(x) = \Phi\left(\frac{x-\mu_c}{\sigma_c}\right) = p' \\ \Rightarrow \quad & x = \mu_c + \sigma_c \Phi^{-1}(p') \\ & \quad = \mu_c + \sigma_c \cdot \sqrt{2} erf^{-1}(2p' - 1), \end{aligned}$$

where $F(x)$ is the cumulative distribution function (CDF), $\Phi(x)$ is the CDF of the standard normal distribution, and $erf^{-1}(z)$ is the *inverse error function* that can be calculated using the Maclaurin series expansion [32].

The value of $ts_{p/2}$ represents the *freshness* of cluster C. We empirically set a freshness threshold as three days before the current time (as empirically no bursty events would last longer) and set $p = 10$. Clusters with $ts_{p/2}$ smaller than the threshold, i.e., the average timestamp of the latest 10 percent tweets is more than three days old, are regarded outdated and removed.

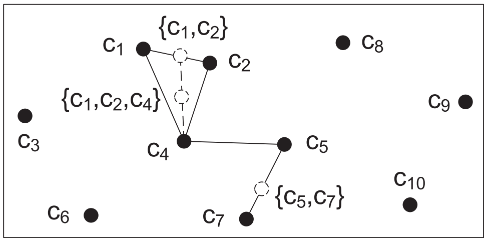


Fig. 4. A running example of cluster merging.

4.1.4 Merging Clusters

If the number of clusters keeps increasing with few deletions, system memory will be exhausted. To avoid this, we specify an upper limit for the number of clusters as N_{max}. When the limit is reached, a merging process starts.

The process merges clusters in a greedy way. First, we sort all cluster pairs by their centroid similarities in a descending order. Then, starting with the most similar pair, we try to merge two clusters in it. When both clusters are *single clusters* which have not been merged with other clusters, they are merged into a new *composite cluster*. When one of them belongs to a composite cluster (it has been merged with others before), the other is also merged into that composite cluster. When both of them have been merged, if they belong to the same composite cluster, this pair is skipped; otherwise, the two composite clusters are merged together. This process continues until there are only mc percentage of the original clusters left (mc is a merging coefficient which provides a balance between available memory space and the quality of remaining clusters).

For cluster merging, each composite cluster is given an *IDList* consisting of *ID*s of the clusters merged in it. And its TCV is obtained by the following operation.

Definition 3 (Aggregation Operation). *Let C_1 and C_2 be two clusters, and their TCVs be $TCV(C_1)$ and $TCV(C_2)$. When C_1 and C_2 are merged together, the composite cluster's $TCV(C_1 \cup C_2)$ is given by*

- $\mathbf{sum\_v} = \mathbf{sum\_v_1} + \mathbf{sum\_v_2}$
- $\mathbf{wsum\_v} = \mathbf{wsum\_v_1} + \mathbf{wsum\_v_2}$
- $ts1 = ts1_1 + ts1_2$
- $ts2 = ts2_1 + ts2_2$
- $n = n_1 + n_2$
- *ft_set consists of the first m tweets in $ft\_set_1 \cup ft\_set_2$, sorted by distance to the centroid of the newly merged cluster.*

Fig. 4 shows a running example of the process. For ease of presentation, we use cluster centroids (the black solid points) to represent clusters and use euclidean distance instead of cosine distance. First, we calculate distances for all cluster pairs and sorted them as $(c_1,c_2),(c_2,c_4),(c_1,c_4),(c_5,c_7),(c_4,c_5)\ldots$. Suppose $mc = 0.7$, then we need to remove $10 \times (1-0.7) = 3$ clusters. To start with, c_1 and c_2 are merged into a composite cluster $\{c_1,c_2\}$. After that, when processing the second pair (c_2,c_4), we find that c_2 has been merged. Hence we combine c_4 into $\{c_1,c_2\}$, and the composite cluster becomes $\{c_1, c_2, c_4\}$. For the next pair (c_1, c_4), since both c_1 and c_4 have been merged, the pair can be skipped. Next we merge c_5 and c_7 into another composite cluster $\{c_5, c_7\}$. Now that we have reduced the number of clusters by 3, the algorithm terminates. Note that since only $N_{max} \times (1 - mc)$ (denoted as N') clusters need to be removed, we would access at most $C^2_{N'} + 1$ pairs.

4.2 High-Level Summarization

The high-level summarization module provides two types of summaries: *online* and *historical* summaries. An online summary describes what is currently discussed among the public. Thus, the input for generating online summaries is retrieved directly from the current clusters maintained in memory.

On the other hand, a historical summary helps people understand the main happenings during a specific period, which means we need to eliminate the influence of tweet contents from the outside of that period. As a result, retrieval of the required information for generating historical summaries is more complicated, and this shall be our focus in the following discussion.

Suppose the length of a user-defined time duration is H, and the ending timestamp of the duration is ts_e. From PTF, we can retrieve two snapshots whose timestamps are either equal to or right before ts_e and $ts_e - H$, respectively. We denote their timestamps by ts_1 and ts_2, and their cluster sets by $S(ts_1)$ and $S(ts_2)$. Now the original duration $[ts_e - H, ts_e]$ is approximated by $[ts_2, ts_1]$.

Intuitively, we need to perform a *cluster set subtraction* between $S(ts_1)$ and $S(ts_2)$. For each cluster C in $S(ts_1)$, we acquire its ID (if it is a single cluster) or IDs in its $IDList$ (a composite cluster). For each of these IDs, we find the corresponding cluster in $S(ts_2)$, and subtract its TCV from C's TCV according to:

Definition 4 (Subtraction Operation). *Given a cluster C_1 in $S(ts_1)$ and its corresponding cluster C_2 in $S(ts_2)$, when C_2 is subtracted from C_1, their difference $TCV(C_1 - C_2)$ is given by*

- $\mathbf{sum\_v} = \mathbf{sum\_v_1} - \mathbf{sum\_v_2}$
- $\mathbf{wsum\_v} = \mathbf{wsum\_v_1} - \mathbf{wsum\_v_2}$
- $ts1 = ts1_1 - ts1_2$
- $ts2 = ts2_1 - ts2_2$
- $n = n_1 - n_2$
- *ft_set consists of tweets which exist in ft_set_1 but not in ft_set_2.*

The above process eliminates the influence of clusters created before ts_2 on summary results. The final set of clusters after this process is the input for historical summarization.

4.2.1 TCV-Rank Summarization Algorithm

Given an input cluster set, we denote its corresponding TCV set as $D(c)$. A tweet set T consists of all the tweets in the ft_sets in $D(c)$. The tweet summarization problem is to extract k tweets from T, so that they can cover as many tweet contents as possible.

Let us first describe this problem formally. Denote $\mathcal{F} = \{T_1, T_2, \ldots, T_t\}$ as a collection of non-empty subsets of T, where a subset T_i represents a sub-topic and $|T_i|$ means the number of its related tweets. Suppose for each T_i, there is a tweet which represents the content of T_i's sub-topic. Then, selecting k tweets is equivalent to selecting k subsets.

Now, the problem can be defined as: given a number k and a collection of sets $\mathcal{F}$, find a subset $\mathcal{F}' \subseteq \mathcal{F}$, such that $|\mathcal{F}'| = k$ and $|\bigcup_{T_i \in \mathcal{F}'} T_i|$ is maximized (i.e., $\mathcal{F}'$ contains as many tweets as possible). We note that this is the *Max-k-Cover* problem, which is NP-hard. Thus, our summarization problem is also NP-hard.

From the geometric interpretation, our summarization tends to select tweets that span the intrinsic subspace of candidate tweet space, such that it can cover most information of the whole tweet set.

Algorithm 2. TCV-Rank Summarization

```
   Input: a cluster set $D(c)$
   Output: a summary set $S$
1  $S = \emptyset, T = \{all\ the\ tweets\ in\ ft\_sets\ of\ D(c)\}$;
2  Build a similarity graph on $T$;
3  Compute LexRank scores $LR$;
4  $T_c = \{tweets\ with\ the\ highest\ LR\ in\ each\ cluster\}$;
5  while $|S| < L$ do
6      foreach tweet $t_i$ in $T_c - S$ do
7          calculate $v_i$ according to Equation (2);
8      select $t_{max}$ with the highest $v_i$;
9      $S.add(t_{max})$;
10 while $|S| < L$ do
11     foreach tweet $t'_i$ in $T - S$ do
12         calculate $v'_i$ according to Equation (2);
13     select $t'_{max}$ with the highest $v'_i$;
14     $S.add(t'_{max})$;
15 return $S$;
```

We design a greedy algorithm to select representative tweets to form summaries (Algorithm 2). First, we build a cosine similarity graph for all the tweets in T (lines 1-2). The maximum size of T is $N \times m$, where N is the number of clusters in $D(c)$ and m is the size of ft_set. It is the upper bound because ft_sets of some clusters (e.g., small clusters or clusters newly created) may not be full. Next, we apply the LexRank method [11] to compute centrality scores for tweets (line 3). LexRank is an effective static summarization method and is efficient for small-sized data sets. But when data sets become large, its efficiency drops quickly (this will be shown in the experiment). The tweet set T has at most Nm tweets (usually hundreds or thousands), so LexRank is suitable for our situation.

However, a potential problem of LexRank is that some top-ranked tweets may have similar contents. Fortunately, since the tweets are retrieved from TCVs, they have got inherent cluster information. Hence, we choose one tweet with the highest LexRank score from each TCV, and add tweet t into the summary (lines 4-9) according to:

$$t = \underset{t_i}{\operatorname{argmax}} \left[\lambda \frac{n_{t_i}}{n_{max}} LR(t_i) - (1 - \lambda) \underset{t_j \in S}{\operatorname{avg}} Sim(t_i, t_j) \right], \tag{2}$$

where λ ($0 \leq \lambda \leq 1$) is a weight parameter, n_{t_i} is the size of the cluster containing t_i, n_{max} is the size of the biggest cluster, $LR(t_i)$ is t_i's LexRank score and S is the summary set containing already chosen tweets.

The motivation of Equation (2) is analogous to that of *maximal marginal relevance (MMR)* [33]. In query-oriented summarization, MMR combines query relevance and information novelty. Here, we combine coverage and novelty as our criterion: the first component on the right side of the equation favors tweets which have high scores and belong to big clusters (content coverage); the second component penalizes redundant tweets with similar contents to those already chosen (novelty).

After the first round selection, if the summary length is still not reached, then we try to select tweets globally ($t_i \in T - S$) based on Equation (2) (lines 10-14).

The computational complexity for LexRank is $O(r|T|^2)$, where r is the iteration number. In tweet selection, note that for each tweet, the first component in the righthand of Equation (2) only needs to be computed once, and the second component can be updated incrementally. So the worst-case cost of tweet selection is $O(N^2 + (|S| - N) \cdot |T|)$. Since $|S| \ll |T|$, the total cost for our algorithm is $O(r|T|^2 + N^2) = O(rm^2N^2)$. As mentioned before, Nm is always controlled at a relatively small number, hence the summarization procedure is very efficient.

4.3 Timeline Generation

The core of the timeline generation module is a topic evolution detection algorithm which produces real-time and range timelines in a similar way. We shall only describe the real-time case here.

The algorithm discovers sub-topic changes by monitoring quantified variations during the course of stream processing. A large variation at a particular moment implies a sub-topic change, which is a new node on the timeline.

The main process is described in Algorithm 3. We first bin the tweets by time (e.g., by day) as the stream proceeds. This sequenced binning is used as input of the algorithm. Then, we loop through the bins and append new timeline nodes whenever large variations are detected (lines 4-5).

Algorithm 3. Topic Evolution Detection

```
Input: a tweet stream binned by time units
Output: a timeline node set $TN$
1: $TN = \emptyset$;
2: while $!stream.end()$ do
3:    $Bin\ C_i = stream.next()$;
4:    if $hasLargeVariation()$ then
5:       $TN.add(i)$;
6: return $TN$;
```

In this algorithm, the key problem is how to define the variation and detect when it becomes large. In what follows, we will describe three different kinds of variations and their detection methods respectively.

4.3.1 Summary-Based Variation

As tweets arrive from the stream, online summaries are produced continuously by utilizing online cluster statistics in TCVs. This allows for generation of a real-time timeline. Generally, when an obvious variation occurs in the main contents discussed in tweets (in the form of summary), we can expect a change of sub-topic (i.e., a time node on the timeline). To quantify the variation, we use the *Jensen-Shannon divergence* to measure the distance between two word distributions in two successive summaries S_c and S_p (S_c is the distribution of the current summary and S_p is that of the previous one)

$$D_{JS}(S_c, S_p) = \frac{1}{2}(D_{KL}(S_c||M) + D_{KL}(S_p||M)). \tag{3}$$

M is the average of the two word distributions, i.e., $M = \frac{1}{2}(S_c + S_p)$. D_{KL} is the *Kullback-Leibler divergence (KLD)* which defines the divergence of distribution M from S

$$D_{KL}(S||M) = \sum_{w \in V} p(w|S) log \frac{p(w|S)}{p(w|M)}. \tag{4}$$

We apply $D_{JS}(S_c, S_p)$ to measure variation instead of directly using $D_{KL}(S_c||S_p)$, because the JSD is a symmetrical and smoothed (the use of $p(w|M)$ avoids zero values of the denominator in Equation (4)) version of the KLD. For ease of comparison, we use the base 2 logarithm in Equation (4), so that the JSD is bounded by $[0, 1]$ [34].

According to this *summary-based variation*, we determine whether the current time is a sub-topic changing node by

$$\frac{D_{cur}}{D_{avg}} > \tau_s, \tag{5}$$

where D_{cur} is the distance between the current summary and its previous neighboring summary, D_{avg} is the average distance of all the previous successive summary pairs which do not produce time nodes, and τ_s ($\tau_s \geq 1$) is the decision threshold.

That is, we detect sub-topic changes whenever there is a *burst* in distances between successive summaries.

4.3.2 Volume-Based Variation

Though the summary-based variation can reflect sub-topic changes, some of them may not be influential enough. Since many tweets are related to users' daily life or trivial events, a sub-topic change detected from textual contents may not be significant enough. To this end, we consider the use of rapid increases (or "spikes") in the volume of tweets over time, which is a common technique in existing online event detection systems [25], [26], [35]. A *spike* suggests that something important just happened because many people found the need to comment on it.

In this part, we develop a spike-finding method. As the input, the binning process in Algorithm 3 needs to count the tweet arrival volume in each time unit. During the stream iteration, the main spike detection logic is:

Heuristic 2. If the tweet volume (C_i) is more than τ_v mean deviations from the mean value, we say that the current moment has a large *volume-based variation*, and identify it as a spike.

$$\frac{C_i - mean}{meandev} > \tau_v. \tag{6}$$

The mean and mean deviation are updated with every new bin count. To adjust to changing tweet frequency trends in a streaming context, we maintain exponentially weighted moving mean and mean deviation (we set $\gamma = 0.125$ as in [25]):

$$\begin{aligned} meandev &= \gamma * |C_i - mean| + (1-\gamma) * meandev, \\ mean &= \gamma * C_i + (1-\gamma) * mean. \end{aligned} \tag{7}$$

4.3.3 Unified Variation

The above spike-finding method may work well for *short-term events* such as football matches, but it would be difficult for them to handle *long-term topic-related streams*, due to some time-aware human behaviors in social media. For instance, the number of tweet posts[2] and public engagement rate[3] will differ in day of week or time of day. Moreover, breaking news or rumors usually draw massive attention and create large spikes in tweet volumes. These spikes will significantly increase the mean and mean deviation values, reducing the chance for subsequent sub-topic changes being detected (Equation (6)).

Since summary- and volume-based methods have their own flaws in detecting topic evolution when they are considered separately, we propose a unified approach combining both factors. Specifically, we define a new quantity called *sum-vol variation*:

$$SV_i = \eta * D^i_{JS}(S_c, S_p) + (1-\eta) * D^i_{vol}, \tag{8}$$

where

$$D^i_{vol} = \begin{cases} 1 - \dfrac{mean}{C_i}, & C_i > mean, \\ 0, & C_i \le mean. \end{cases}$$

The first part on the right side of Equation (8) is the summary-based component and the second part is the volume-based component normalized into $[0, 1)$. η is the balance factor between two components.

Now we can decide whether the current time is a sub-topic changing node by

$$\frac{SV_{cur}}{SV_{avg}} > \tau_{sv}, \tag{9}$$

where SV_{cur} is the sum-vol variation of the current moment, SV_{avg} is the average variation of all the previous moments and is updated in the same way as Equation (7).

In this way, we take both effects of content (summaries) and significance (tweet volumes) into consideration, and detect topic evolution whenever there is a burst in the sum-vol variation.

4.4 Discussion

Handling noises. The effect of clusters of noises can be diminished by two means in Sumblr. First, in tweet stream clustering, noise clusters which are not updated frequently will be deleted as outdated clusters. Second, in the summarization step, tweets from noise clusters are far less likely to be selected into summary, due to their small LexRank scores and cluster sizes.

TABLE 2
Basic Information of Data Sets

Topics (filtering keywords)	#Tweets	Time Span
Obama	95,055	2009.02 - 2009.10
Chelsea	438,884	2012.11 - 2012.12
Arsenal Arsene Wenger	323,555	2012.11 - 2012.12
Tablet Smartphone Cellphone	231,011	2012.11 - 2012.12
Black Friday	124,684	2012.11 - 2012.12

Extension to multi-topic streams. So far we have assumed a tweet stream of only one topic as the input to Sumblr. However, we should note that Sumblr can be easily extended for multi-topic streams. For example, when a new tweet arrives, we first decide its related topics by keyword matching. Then it is delivered into different groups of clusters. Clusters are grouped by their corresponding topical IDs. Consequently, Sumblr is applied within each cluster group. It is important to note that this mechanism allows for distributed system implementation.

5 Experiments

In this section, we evaluate the performance of Sumblr. We present the experiments for summarization and timeline generation respectively.

5.1 Experiments for Summarization

5.1.1 Setup

Data Sets. We construct five data sets to evaluate summarization. One is obtained by conducting keyword filtering on a large Twitter data set used by [31]. The other four include tweets acquired during one month in 2012 via Twitter's keyword tracking API.[4] As we do not have access to the respective users' social networks for these four, we set their weights of tweets w_i to the default value of 1. Details of the data sets are listed in Table 2.

Ground truth for summaries. As no previous work has conducted similar study on continuous summarization, we have to build our own ground truth (reference summaries). However, manual creation of these summaries is apparently impractical due to the large size of the data sets.

Thus, we employ a two-step method to obtain fair-quality reference summaries:

1) Given a time duration, we first retrieve the corresponding tweet subset, and use the following three well-recognized summarization algorithms to get three candidate summaries.

ClusterSum [16] clusters the tweets and picks the most weighted tweet from each cluster to form summary.

LexRank [11] first builds a sentence similarity graph, and then selects important sentences based on the concept of eigenvector centrality.

DSDR [13] models the relationship among sentences using linear reconstruction, and finds an optimal set of sentences to approximate the original documents, by minimizing the reconstruction error.

[2] http://www.sysomos.com/insidetwitter/

[3] http://www.mediapost.com/publications/article/208671/which-day-will-your-facebook-brand-post-work-best.html

[4] https://dev.twitter.com/docs/api/1.1/post/statuses/filter

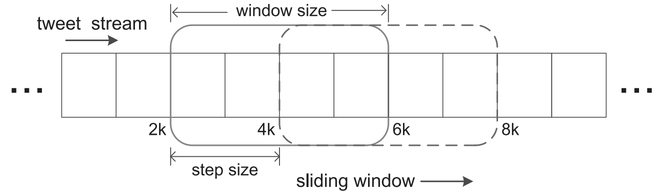


Fig. 5. The sliding window mechanism.

2) Next, for each subset, the final reference summary is extracted from three candidate summaries by using a voting scheme. The intuition is that if a specific tweet and its similar tweets appear many times in the candidate summaries, they can represent important content and should be chosen into the final summary. Specifically, for each tweet in each candidate summary, we compute its similarities to the tweets from the other two candidate summaries. Then, the tweet votes to its most similar one and these two tweets form a "pair". After processing all the tweets in candidate summaries, we sort them in descending order of their total votes. We delete those tweets whose pair members already exist at higher ranks. At last, the top ranking tweets are added into the final reference summary until the summary length is reached.

Baseline methods. Existing summarization methods have not been designed to handle continuous summarization. However, they can be adapted to streaming data by using a *sliding window* scheme. As illustrated in Fig. 5, each window contains a certain number (window size) of tweets which are summarized as a document. After that, the window moves forward by a *step size*, so that the oldest tweets are discarded and the new ones are added into the window. In this way, we implement the sliding window version of the above three algorithms, namely *ClusterSum*, *LexRank*, and *DSDR*. The windows are dimensioned by number of tweets instead of time duration, because the number of tweets may vary dramatically across fixed-length durations, leading to very poor performance of the baseline algorithms.

Evaluation method. We apply the popular ROUGE toolkit [36] for evaluation. Among supported metrics, ROUGE-1 has been demonstrated to be the most consistent with human judgement [37]. Considering the short and informal nature of the tweet contents, we decide that ROUGE-1 is suitable for measuring the quality of tweet summaries.

To evaluate the metric on a continuous time period $[T_0, T]$, we have to calculate the integral of the metric over the period, which is given by

$$\int_{T_0}^{T} metric(t)\, dt. \tag{10}$$

The integral requires unaffordable number of samplings during the period. In practice, we only sample the metrics by each arrival of a certain number of new tweets. This number is called the *sampling interval*. Note the sampling interval must be no greater than the step size.

In our experiments, we find similar trends in the comparison of precision, recall and F-score between the proposed approach and the baseline methods. Therefore, we shall only report the F-score results to save space. The F-score results presented are averaged on all five data sets.

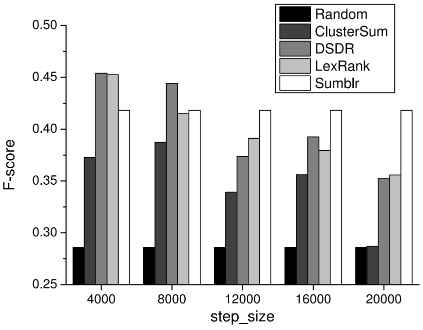


Fig. 6. Quality on step size.

5.1.2 Overall Performance Comparison

In this section, we compare the F-scores and runtime costs between Sumblr and three baseline algorithms (sliding window version). As tweets are often produced very quickly and reach a huge volume in a short while, it is hardly meaningful to summarize a small number of tweets. Thus the window size should be a relatively large one. In this experiment, we set window size to 20,000 and sampling interval to 2,000. The step size varies from 4,000 to 20,000. The metrics are averaged over the whole stream.

Figs. 6 and 7 present the results for different step sizes. In Fig. 6, we also give a baseline *Random* method, which selects tweets randomly from each window. Note that Sumblr is not affected by the step size, as it supports continuous summarization inherently.

The results show that when the step size is small (4,000), DSDR and LexRank achieve better summary quality than Sumblr, at the expense of much more computation. When $step\_size \geq 12{,}000$, Sumblr outperforms all the baseline methods in terms of both summary quality and computation cost. Although the efficiency of LexRank is comparable with our method when $step\_size \geq 16{,}000$, its summary quality is significantly worse.

The above results reveal a major problem with the baseline methods: they rely on a small step size to produce quality summaries. Unfortunately, a small step size leads to very frequent and expensive computations for windows. In contrast, Sumblr strikes a good balance between summary quality and efficiency.

Another issue to note is that, as the ground truth is generated using these baseline methods, the summary quality is to some extent biased in favor of them.

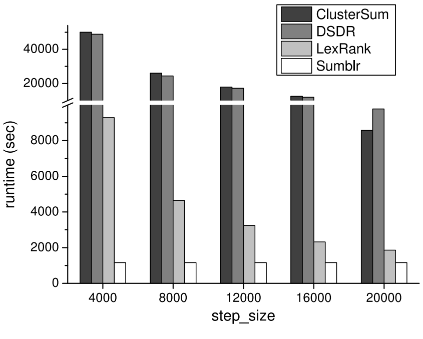


Fig. 7. Efficiency on step size.

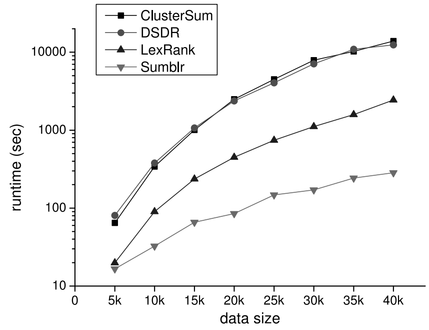


Fig. 8. Scalability on data size.

Scalability. The scalability experiment evaluates the efficiency results of a single window, while varying the window size. It simulates the case of a large burst of tweets in a short period.

Fig. 8 presents the scalability results for different methods. Note that the y-axis is in the log scale. We can see that our method outperforms the others significantly. When the data size is above 15,000, Sumblr is faster than LexRank by nearly an order of magnitude, and outperforms the other two by more than that. The small fluctuations in Sumblr may be caused by the cluster deletion and merging operations.

5.1.3 Parameter Tuning

In this section, we tune the parameters in our approach. In each of the following experiments, we vary one parameter and keep the others fixed.

Effect of β. In Heuristic 1 we use β to determine whether to create a new cluster. Figs. 9a and 9b show its effect on summary quality and efficiency. When β is small, tweets related to different sub-topics may be absorbed into the same clusters, so the input of our summarization component is of low quality. At the same time, there are many focus tweets in each cluster, thus the time cost of cluster updating and summarization is high. When β increases, too many clusters are created, causing damage to both quality and efficiency. A good choice is $\beta = 0.07$ as it gives more balanced results.

Effect of N_{max}. Figs. 9c and 9d depict the performance of N_{max}. For small N_{max}s, many merging operations are conducted, which are time-consuming and produce lots of low-quality clusters. For large values, stream clustering is slow due to large number of clusters. Note that the storage overhead (both in memory and disk) is also higher for larger N_{max}s. A balanced value for N_{max} is 150.

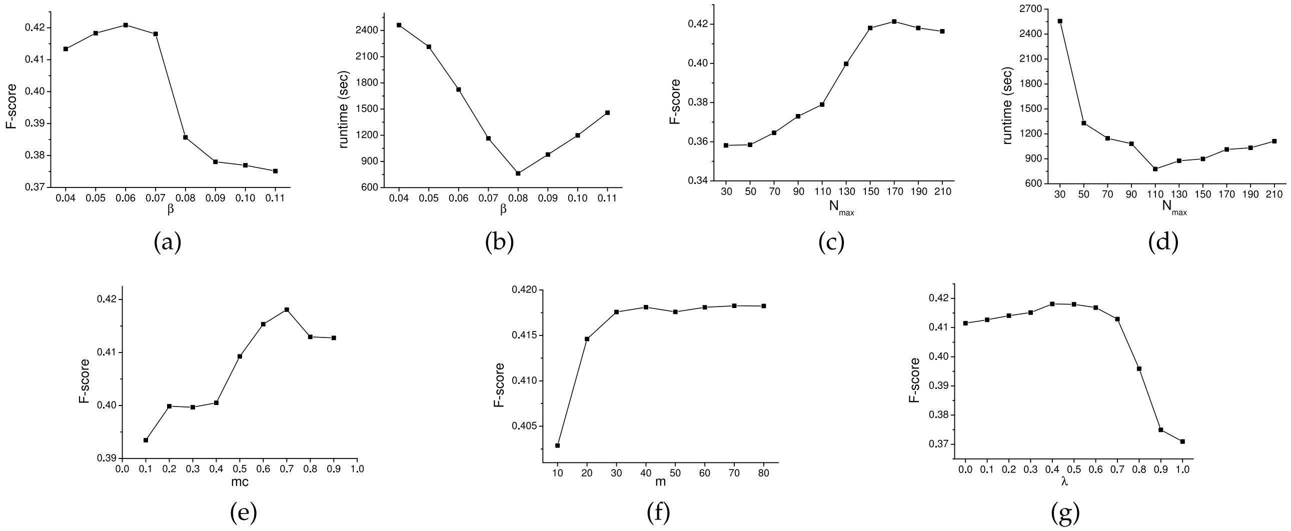
Effect of mc. Another parameter in cluster merging is mc $(0 < mc < 1)$. It does not have significant impact on efficiency, so we only present its quality results (Fig. 9e). Small values of mc result in low-quality clusters, while large ones lead to many merging operations, which in turn reduce the quality of clusters. An ideal value for mc is 0.7.

Effect of m. As shown in Fig. 9f, the summary quality improves when m increases. When $m \geq 40$, the improvement is not obvious. Meanwhile, a larger m incurs more storage overhead. We choose $m = 40$.

Effect of λ. Finally we check the effect of λ $(0 \leq \lambda \leq 1)$, which is a tradeoff factor between content coverage and novelty. We gradually vary λ from 0 to 1 at the step of 0.1 to examine its effect, as shown in Fig. 9g. When $\lambda \geq 0.7$, the extreme emphasis on coverage causes performance loss. Therefore, we set $\lambda = 0.4$ as a balanced factor.

5.1.4 Flexibility

One distinguishing feature of Sumblr is the flexibility to summarize tweets over arbitrary time durations. This feature is provided by incorporating the PTF. The effectiveness of PTF depends on α and l (Section 3.3). We fix α at 2 and show the results varying l. For consistency, we extract a subset of one-month period from each data set as the input stream. The interval between two successive snapshots (timestamp unit) is one hour. For a timestamp ts, we evaluate the results for different durations with length len varying from 1 to 10 days. We report the average F-score $score(ts)$ by interval of 48 hours. Due to space limit, we only



(a) (b) (c) (d)

(e) (f) (g)

Fig. 9. Performance of different parameters. (a) Effect of β on quality. (b) Effect of β on efficiency. (c) Effect of N_{max} on quality. (d) Effect of N_{max} on efficiency. (e) Effect of mc. (f) Effect of m. (g) Effect of λ.

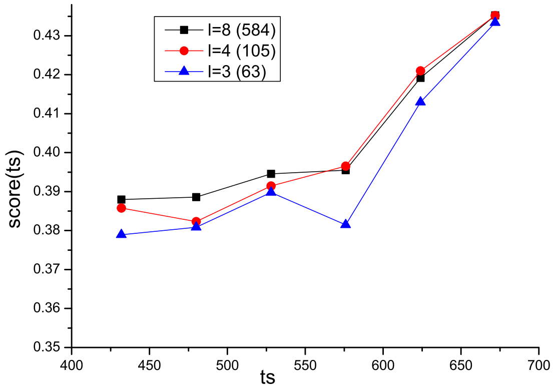


Fig. 10. Quality on time duration.

show the figure for "Arsenal", results for other data sets are available on the web.[5]

$$score(ts) = \frac{\sum_{len} F\text{-}score([ts - len, ts])}{10}. \qquad (11)$$

Fig. 10 gives the following observations:

- There exists a common trend: more recent time durations have higher summary quality. This is because PTF has finer granularity of snapshots for more recent moments. As a result, the queried durations can be better approximated.
- A larger l leads to higher overall quality. Due to larger capacity of each order, PTF with a larger l is able to maintain more snapshots, and thus produce more accurate approximation for the queried durations.
- Unfortunately, a larger l also requires more storage cost (the numbers in the parentheses represent the amounts of snapshots in PTF). This is obvious since it enables PTF to store more snapshots, which results in heavier storage burden.

For different applications, Sumblr can be customized with different l values. For example, for real-time summarization, a small l is enough; while for historical review, a large l is needed.

Granularity. To further evaluate the flexibility of Sumblr, we also conduct a granularity test. We partition the one-month data sets into time durations with fixed length (e.g., 24, 72, or 144 hours), then report the average F-scores for these durations under different levels of granularity. As shown in Table 3, summary quality does not have significant difference among different granularities. This is because the quality of a historical summary mainly depends on two endpoints of the duration (i.e., the accuracy of duration approximation in PTF) rather than the length of the duration.

5.2 Experiments for Timeline Generation

5.2.1 Data Sets and Ground Truth

In this section, we evaluate the effectiveness of topic evolution detection, i.e., timeline generation. We use the "Arsenal" and "Chelsea" data sets in Section 5.1, and add two more recent data sets ("Arsenal2013" and "Chelsea2013"). We choose these data sets because reference timelines for sport topics are relatively easier to build. For this experiment, the reference timelines are manually produced. Specifically, we read through all the related news during those periods of the corresponding data sets from news websites (Yahoo!, ESPN, etc.), and select those *dates* as nodes on the reference timelines when something important happen, e.g., football matches, players' signing of new contracts, etc. The time unit of timelines is day. Statistics of the data sets and the reference timelines (numbers of timeline nodes) are listed in Table 4.

TABLE 3
Results of Different Granularities ($\alpha = 2, l = 6$)

	24	72	144
Obama	0.4567	0.4867	0.4765
Chelsea	0.3089	0.3474	0.3019
Arsenal	0.4197	0.4116	0.3876
Smartphone	0.4214	0.3855	0.4017
Black Friday	0.3275	0.3613	0.3537

5.2.2 Results

Our objective is to detect nodes in the reference timeline as the stream proceeds. We compare performance of the topic evolution detection algorithm using three different variations in Section 4.3, i.e., summary-based variation, volume-based variation and sum-vol variation. We present precision, recall, and F-score of the timeline nodes detected by these methods. Since similar trends are observed in all four data sets, here we only show results for the largest data set Chelsea2013 to save space. Results for other data sets are also available.[5]

Figs. 11 and 12 show the effects of the decision threshold for SUM (τ_s) and VOL (τ_v), respectively. In both figures, as the threshold increases, recall declines while precision increases. This is expected since higher threshold would exclude more promising candidate nodes, and those remaining nodes with larger variations are more likely to be the correct ones.

We get the highest F-scores when $\tau_s = 1.10$ for SUM and $\tau_v = 1.05$ for VOL. Note that SUM has relatively higher recall while VOL has higher precision. This is also consistent with our analysis in Section 4.3: SUM detects sub-topic changes based on content variation, but some of them may not be influential enough; VOL finds spikes when lots of people talk about something important, but reduces the chance for other sub-topic changes being detected. In other words, we can consider SUM as a "sensitive" method and VOL as a "conservative" method.

Next, we present effects of the unified sum-vol variation. Fig. 14 shows the F-score results under different

TABLE 4
Statistics of Data Sets

Data Sets	#Tweets	Time Span	#Timeline Nodes
Arsenal2012	323,555	2012.11 - 2012.12	10
Chelsea2012	438,884	2012.11 - 2012.12	9
Arsenal2013	606,698	2013.03 - 2013.05	12
Chelsea2013	899,287	2013.03 - 2013.06	20

[5] http://db.zju.edu.cn/s/sumblr/ext.pdf

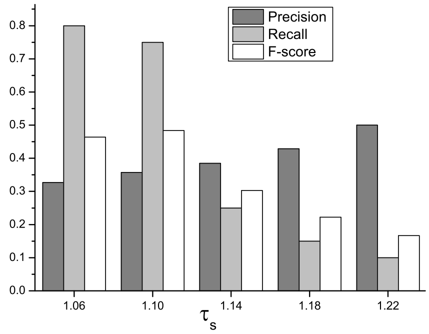


Fig. 11. Effect of τ_s (SUM).

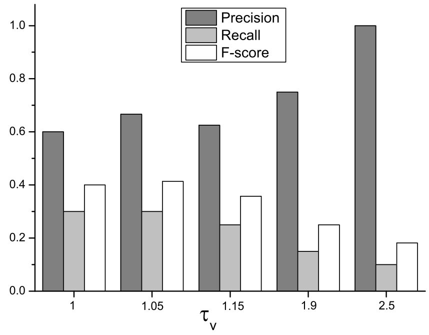


Fig. 12. Effect of τ_v (VOL).

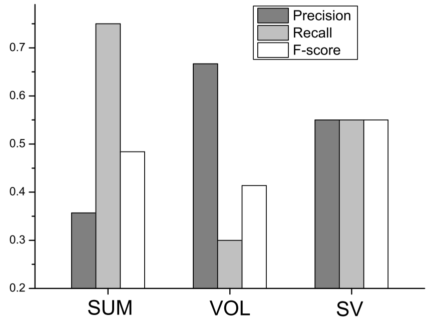


Fig. 13. Comparison of three methods.

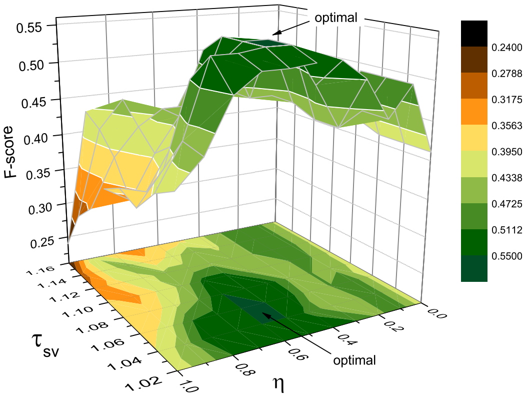


Fig. 14. Effect of τ_{sv} and η on F-score (SV).

settings of τ_{sv} and η. The optimal point are indicated in the figure, where $\tau_{sv} = 1.06$ and $\eta = 0.6$. We also draw a contour at the bottom, which illustrates the effect of the combination parameter η. When τ_{sv} is at the optimal value (about 1.06), F-score reaches the highest value at $\eta \in [0.5, 0.6]$ (i.e., summary-based and volume-based information almost equally contribute to the sum-vol variation).

In Fig. 13, we compare SV with SUM and VOL, all with their optimal F-score settings. By integrating content and significance factors into a unified detection mechanism, SV significantly improves the F-score (about 14 percent to SUM and 33 percent to VOL) while maintaining balanced precision and recall.

Finally, we present output timelines of the SV approach. Each timeline consists of five nodes, and each node contains a timestamp and a summary. The nodes whose timestamps are included in braces are contained in the reference timelines. Due to space limit, we only show five nodes of each timeline and two sentences in the summary of each node.

TABLE 5
Selected Part of Timelines for "Arsenal2012" (Left) and "Chelsea2013" (Right)

Arsenal2012	Chelsea2013
{12.09}: 1. Arsenal 1-0 West Brom! Come on Arsenal! 2. Come on @Arsenal ♯MustWin	**{03.15}**: 1. GOAL: CHELSEA 3-1 Steaua Bucharest - Torres (71) 2. Chelsea defeat Steaua Bucharest 3-1 to claim a 3-2 aggregate win and a last-eight place in the Europa League.
{12.12}: 1. Yes Bradford! WENGER OUT. 2. And Arsene Wenger thinks this ♯Arsenal team can still win the league and or finish in the top 4!!! They can't even beat Bradford!!	**03.17**: 1. Chelsea v westham tomoz there are soo many players injured at westham :'(2. Chelsea vs West Ham! GO West Ham!!!
{12.18}: 1. Arsenal vs Reading Come on arsenal ♯Arsenal 2. What a come back! Cazorla is a class! But I love how Wenger gave the right position for Walcott.	**{03.18}**: 1. Lampard scores his 200th Chelsea goal. Chelsea 1-0 West Ham. 2. (Sky Sport) Zola quiet on Chelsea rumours.
{12.19}: 1. Jack Wilshere, Alex Oxlade-Chamberlain, Aaron Ramsey, Kieran Gibbs and Carl Jenkinson sign long-term deals at Arsenal. No Theo though ♯afc 2. Arsenal vs West Ham Boxing Day match off due to Tube strike.	**{03.31}**: 1. HT : Chelsea 1 - 2 Southampton Come On CHELSEA !!! 2. Southampton 2-1 CHELSEA... Southampton 3-1 City... Southampton 3-1 Liverpool... Southampton 2-3 MAN UNITED!! WE ARE UNITED!!
{12.20}: 1. Champions League draw: Manchester United v Real Madrid, Arsenal v Bayern Munich, Celtic v Juventus. 2. Arsenal vs Bayern Munich gonna be a tough game.	**{04.01}**: 1. Chelsea v Man u. No Manchester United please. Come on Chelsea! 2. Chelsea defeats Man United 1-0, stay alive in FA Cup. Chelsea haven't lost an FA Cup game at home in an incredible 10 YEARS.

Timelines for “Arsenal2012” and “Chelsea2013” are demonstrated in Table 5. It is interesting to find that our approach detects not only important events (e.g., on 18 March 2013, Lampard scored his 200th Chelsea goal), but also popular public opinions (e.g., on 12 December 2012, public calling for WENGER OUT).

6 Conclusion

We proposed a prototype called Sumblr which supported continuous tweet stream summarization. Sumblr employs a tweet stream clustering algorithm to compress tweets into TCVs and maintains them in an online fashion. Then, it uses a TCV-Rank summarization algorithm for generating online summaries and historical summaries with arbitrary time durations. The topic evolution can be detected automatically, allowing Sumblr to produce dynamic timelines for tweet streams. The experimental results demonstrate the efficiency and effectiveness of our method. For future work, we aim to develop a multi-topic version of Sumblr in a distributed system, and evaluate it on more complete and large-scale data sets.

Acknowledgments

The work was supported by the National Science Foundation of China (Grant No. 61170034) and the National High Technology Research and Development Program of China (Grant No. 2013AA040601).

References

[1] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu, “A framework for clustering evolving data streams,” in *Proc. 29th Int. Conf. Very Large Data Bases*, 2003, pp. 81–92.

[2] T. Zhang, R. Ramakrishnan, and M. Livny, “BIRCH: An efficient data clustering method for very large databases,” in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 1996, pp. 103–114.

[3] P. S. Bradley, U. M. Fayyad, and C. Reina, “Scaling clustering algorithms to large databases,” in *Proc. Knowl. Discovery Data Mining*, 1998, pp. 9–15.

[4] L. Gong, J. Zeng, and S. Zhang, “Text stream clustering algorithm based on adaptive feature selection,” *Expert Syst. Appl.*, vol. 38, no. 3, pp. 1393–1399, 2011.

[5] Q. He, K. Chang, E.-P. Lim, and J. Zhang, “Bursty feature representation for clustering text streams,” in *Proc. SIAM Int. Conf. Data Mining*, 2007, pp. 491–496.

[6] J. Zhang, Z. Ghahramani, and Y. Yang, “A probabilistic model for online document clustering with application to novelty detection,” in *Proc. Adv. Neural Inf. Process. Syst.*, 2004, pp. 1617–1624.

[7] S. Zhong, “Efficient streaming text clustering,” *Neural Netw.*, vol. 18, nos. 5/6, pp. 790–798, 2005.

[8] C. C. Aggarwal and P. S. Yu, “On clustering massive text and categorical data streams,” *Knowl. Inf. Syst.*, vol. 24, no. 2, pp. 171–196, 2010.

[9] R. Barzilay and M. Elhadad, “Using lexical chains for text summarization,” in *Proc. ACL Workshop Intell. Scalable Text Summarization*, 1997, pp. 10–17.

[10] W.-T. Yih, J. Goodman, L. Vanderwende, and H. Suzuki, “Multi-document summarization by maximizing informative content-words,” in *Proc. 20th Int. Joint Conf. Artif. Intell.*, 2007, pp. 1776–1782.

[11] G. Erkan and D. R. Radev, “LexRank: Graph-based lexical centrality as salience in text summarization,” *J. Artif. Int. Res.*, vol. 22, no. 1, pp. 457–479, 2004.

[12] D. Wang, T. Li, S. Zhu, and C. Ding, “Multi-document summarization via sentence-level semantic analysis and symmetric matrix factorization,” in *Proc. 31st Annu. Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, 2008, pp. 307–314.

[13] Z. He, C. Chen, J. Bu, C. Wang, L. Zhang, D. Cai, and X. He, “Document summarization based on data reconstruction,” in *Proc. 26th AAAI Conf. Artif. Intell.*, 2012, pp. 620–626.

[14] J. Xu, D. V. Kalashnikov, and S. Mehrotra, “Efficient summarization framework for multi-attribute uncertain data,” in *Proc. ACM SIGMOD Int. Conf. Manage.*, 2014, pp. 421–432.

[15] B. Sharifi, M.-A. Hutton, and J. Kalita, “Summarizing microblogs automatically,” in *Proc. Human Lang. Technol. Annu. Conf. North Amer. Chapter Assoc. Comput. Linguistics*, 2010, pp. 685–688.

[16] D. Inouye and J. K. Kalita, “Comparing twitter summarization algorithms for multiple post summaries,” in *Proc. IEEE 3rd Int. Conf. Social Comput.*, 2011, pp. 298–306.

[17] S. M. Harabagiu and A. Hickl, “Relevance modeling for microblog summarization,” in *Proc. 5th Int. Conf. Weblogs Social Media*, 2011, pp. 514–517.

[18] H. Takamura, H. Yokono, and M. Okumura, “Summarizing a document stream,” in *Proc. 33rd Eur. Conf. Adv. Inf. Retrieval*, 2011, pp. 177–188.

[19] C. Shen, F. Liu, F. Weng, and T. Li, “A participant-based approach for event summarization using twitter streams,” in *Proc. Human Lang. Technol. Annu. Conf. North Amer. Chapter Assoc. Comput. Linguistics*, 2013, pp. 1152–1162.

[20] D. Chakrabarti and K. Punera, “Event summarization using tweets,” in *Proc. 5th Int. Conf. Weblogs Social Media*, 2011, pp. 66–73.

[21] M. Kubo, R. Sasano, H. Takamura, and M. Okumura, “Generating live sports updates from twitter by finding good reporters,” in *Proc. IEEE Int. Joint Conf. Web Intell. Agent Technol.*, 2013, pp. 527–534.

[22] N. A. Diakopoulos and D. A. Shamma, “Characterizing debate performance via aggregated twitter sentiment,” in *Proc. SIGCHI Conf. Human Factors Comput. Syst.*, 2010, pp. 1195–1198.

[23] M. Dork, D. Gruen, C. Williamson, and S. Carpendale, “A visual backchannel for large-scale events,” *IEEE Trans. Vis. Comput. Graph.*, vol. 16, no. 6, pp. 1129–1138, Nov. 2010.

[24] R. Yan, X. Wan, J. Otterbacher, L. Kong, X. Li, and Y. Zhang, “Evolutionary timeline summarization: A balanced optimization framework via iterative substitution,” in *Proc. 34th Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, 2011, pp. 745–754.

[25] A. Marcus, M. S. Bernstein, O. Badar, D. R. Karger, S. Madden, and R. C. Miller, “Twitinfo: Aggregating and visualizing microblogs for event exploration,” in *Proc. SIGCHI Conf. Human Factors Comput. Syst.*, 2011, pp. 227–236.

[26] J. Nichols, J. Mahmud, and C. Drews, “Summarizing sporting events using twitter,” in *Proc. ACM Int. Conf. Intell. User Interfaces*, 2012, pp. 189–198.

[27] L. Hong, A. Ahmed, S. Gurumurthy, A. J. Smola, and K. Tsioutsiouliklis, “Discovering geographical topics in the twitter stream,” in *Proc. 21st Int. Conf. World Wide Web*, 2012, pp. 769–778.

[28] C. Lin, C. Lin, J. Li, D. Wang, Y. Chen, and T. Li, “Generating event storylines from microblogs,” in *Proc. 21st ACM Int. Conf. Inf. Knowl. Manage.*, 2012, pp. 175–184.

[29] X. Yang, A. Ghoting, Y. Ruan, and S. Parthasarathy, “A framework for summarizing and analyzing twitter feeds,” in *Proc. Knowl. Discovery Data Mining*, 2012, pp. 370–378.

[30] B. Van Durme, “Streaming analysis of discourse participants,” in *Proc. Joint Conf. Empirical Methods Natural Lang. Process. Comput. Natural Lang. Learning*, 2012, pp. 48–58.

[31] C. Chen, F. Li, B. C. Ooi, and S. Wu, “TI: An efficient indexing mechanism for real-time search on tweets,” in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2011, pp. 649–660.

[32] D. Zwillinger, *CRC Standard Mathematical Tables and Formulae*. Boca Raton, FL, USA: CRC Press, 2011.

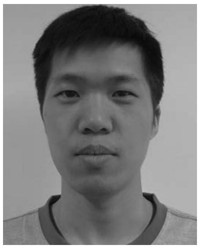
[33] J. Carbonell and J. Goldstein, “The use of MMR, diversity-based reranking for reordering documents and producing summaries,” in *Proc. 21st Annu. Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, 1998, pp. 335–336.

[34] J. Lin, “Divergence measures based on the shannon entropy,” *IEEE Trans. Inf. Theory*, vol. 37, no. 1, pp. 145–151, Jan. 1991.

[35] A. Zubiaga, D. Spina, E. Amigó, and J. Gonzalo, “Towards real-time summarization of scheduled events from twitter streams,” in *Proc. 23rd ACM Conf. Hypertext Social Media*, 2012, pp. 319–320.

[36] C.-Y. Lin, “Rouge: A package for automatic evaluation of summaries,” in *Proc. ACL Workshop Text Summarization Branches Out*, 2004, pp. 74–81.

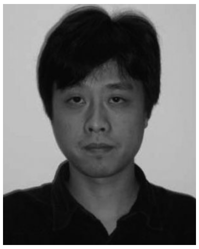
[37] C.-Y. Lin and E. Hovy, “Automatic evaluation of summaries using n-gram co-occurrence statistics,” in *Proc. Human Lang. Technol. Annu. Conf. North Amer. Chapter Assoc. Comput. Linguistics*, 2003, pp. 71–78.



Zhenhua Wang received the BS degree in computer science from Zhejiang University in 2008, and is currently working toward the PhD degree in the College of Computer Science, Zhejiang University. His research interests include spatial database and web data mining.

Lidan Shou received the PhD degree in computer science from the National University of Singapore. He is currently a professor with the College of Computer Science, Zhejiang University, China. Prior to joining the faculty, he worked in the software industry for more than two years. His research interests include spatial database, data access methods, visual and multimedia databases, and web data mining. He is a member of the ACM.

Ke Chen received the PhD degree in computer science in 2007, and later she went on to become a postdoctoral fellow in the School of Aeronautics and Astronautics of Zhejiang University till 2009. She is currently an associate professor in the College of Computer Science, Zhejiang University. Her research interests include spatial temporal data management, web data mining, and data privacy protection. She is a member of the ACM.



Gang Chen received the PhD degree in computer science from Zhejiang University. He is a professor in the College of Computer Science and the director of the Database Lab, Zhejiang University. He has successfully led the investigation in research projects which aim at building China's indigenous database management systems. His research interests range from relational database systems to large-scale data management technologies supporting massive Internet users. He is a member of the ACM and a senior member of the China Computer Federation.

Sharad Mehrotra is a professor in the School of Information and Computer Science at the University of California, Irvine, and the founding director of the Center for Emergency Response Technologies (CERT) at UCI. His research expertise is in data management and distributed systems areas in which he has made many pioneering contributions. Two such contributions include the concept of "use of information retrieval techniques, particularly relevance feedback, in multimedia search" and "database as a service." He received the ACM SIGMOD Test of Time Award in 2012 and DASFAA 10 year Best Paper Awards for the years 2013 and 2014. In addition, several of his papers have received the Best Paper nominations and awards including SIGMOD Best Paper Award in 2001, Best Paper Award in DASFAA 2004, and the Best Paper Award nomination in the ACM International conference in Multimedia Retrieval, 2013. His current research interests include data quality, privacy, and big data analytics.