Sumblr: Continuous Summarization of Evolving Tweet Streams

Lidan Shou [1,2] Zhenhua Wang [1] Ke Chen[1] Gang Chen[1]
[1]College of Computer Science and Technology
[2]State Key Lab of CAD&CG
Zhejiang University
Hangzhou, China
{should, wzh-cs, chenk, cg}@zju.edu.cn

ABSTRACT

With the explosive growth of microblogging services, short-text messages (also known as tweets) are being created and shared at an unprecedented rate. Tweets in its raw form can be incredibly informative, but also overwhelming. For both end-users and data analysts it is a nightmare to plow through millions of tweets which contain enormous noises and redundancies. In this paper, we study continuous tweet summarization as a solution to address this problem. While traditional document summarization methods focus on static and small-scale data, we aim to deal with dynamic, quickly arriving, and large-scale tweet streams. We propose a novel prototype called Sumblr (SUMmarization By stream cLusteRing) for tweet streams. We first propose an online tweet stream clustering algorithm to cluster tweets and maintain distilled statistics called Tweet Cluster Vectors. Then we develop a TCV-Rank summarization technique for generating online summaries and historical summaries of arbitrary time durations. Finally, we describe a topic evolvement detection method, which consumes online and historical summaries to produce timelines automatically from tweet streams. Our experiments on large-scale real tweets demonstrate the efficiency and effectiveness of our approach.

Categories and Subject Descriptors

H.3.1 [**Content Analysis and Indexing**]: Abstracting methods; H.3.4 [**Systems and Software**]: Performance evaluation (efficiency and effectiveness)

Keywords

Tweet stream; continuous summarization; timeline

1. INTRODUCTION

With the explosive growth of microblogging services, such as Twitter, Weibo and Tumblr, short-text messages known as tweets are being created and shared at an unprecedented rate. Take Twitter for instance, receiving over 400 million tweets per day[1], it has become an invaluable source of news, blogs, opinions, and more. Tweets in its raw form can be incredibly informative, but also overwhelming. For example, searching for a hot topic in Twitter can yield millions of tweets, which span for weeks. Even if filtering is allowed, plowing through so many tweets for interesting contents would be a nightmare, not to mention the enormous noises and redundancies that one could encounter. To make things worse, new tweets satisfying the filtering criteria may arrive continuously, at an unpredictable rate.

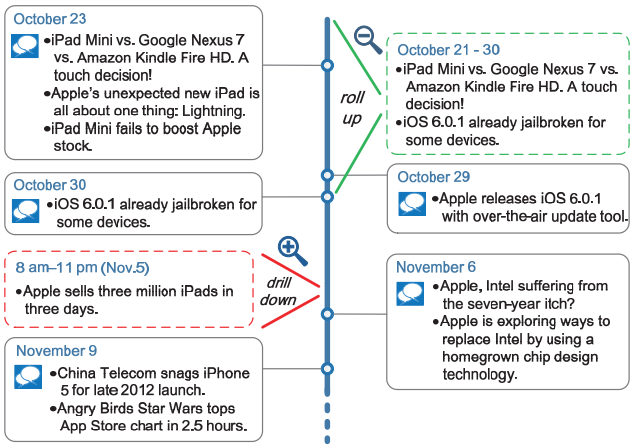


Figure 1: A timeline example for topic "Apple"

A possible solution to the above problem is *continuous tweet summarization*, which represents the massive tweets in a set of short text pieces covering the main topics (or sub-topics of course). Specifically, let us look at an example, which presumes availability of a topic-related tweet stream, for example tweets about "Apple". With a tweet summarization system, we can (i) continuously monitor "Apple"-related tweets arriving from the stream and produce a *continuous timeline* which grows by time. (ii) Suppose a user wants to learn the main happenings about "Apple" from the tweets between 22 Oct 2012 and 11 Nov 2012. The *range timeline* during that period can be provided to her, so that she understands the big picture of topic evolution in those weeks (as shown in Figure 1). (iii) After that, she may need more detailed reports for a much smaller duration (e.g. from 8 am to 11 pm on 5 Nov), which is like a *drill-down summary* on the duration. (iv) Alternatively, she may ask for a more concise report during 21 Oct to 30 Oct, in a *roll-up summary*.

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

SIGIR'13, July 28–August 1, 2013, Dublin, Ireland.

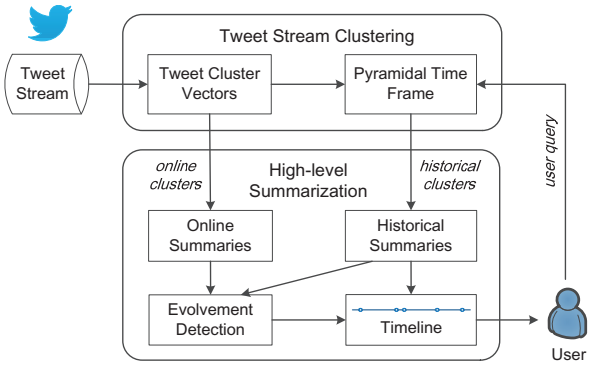


Figure 2: The framework of Sumblr

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.

Implementing continuous tweet stream summarization is not an easy task, as the tweets are of noisy, redundant, and social nature. More importantly, tweets arrive very quickly and are strongly correlated with their posted time. A good solution to continuous tweet stream summarization has to address the following issues: (1) *Efficiency* - tweet streams always have very large scales, so their summarization should be highly efficient, with only one pass over the data; (2) *Flexibility* - the ability to provide tweet summaries of arbitrary time durations. (3) *Topic evolvement* - to automatically detect sub-topic changes and the moments that they happen.

Unfortunately, the importance of continuous summarization has long been overlooked by the research community. Although there exist numerous studies on document summarization [6, 26, 23, 13, 9, 11], these methods cannot satisfy our requirements, because: (1) They mainly focus on static and small-sized datasets, making it intractable to improve their efficiency. (2) To provide summary for arbitrary duration, these techniques will have to perform iterative/recursive summarization for every possible time duration, which is unacceptable. (3) The summary results of these algorithms are insensitive to time. Thus it is difficult for them to detect topic evolvement.

In this paper, we introduce a novel tweet summarization prototype called *Sumblr (SUMmarization By stream cLusteRing)*. To the best of our knowledge, our work is the first to study continuous tweet stream summarization. The overall structure of the prototype is depicted in Figure 2. Sumblr consists of two main components, namely a *Tweet Stream Clustering* module and a *High-level Summarization* 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 uses two data structures to keep important tweet information in clusters. The first one is a compressed structure called *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 the generation of two kinds of summaries: online summaries and historical ones. (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 historical summaries 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.

The summarization module also contains a topic evolvement detection algorithm, which consumes online/historical summaries to produce continuous/range timelines. A timeline is a sequence of time-stamped summaries (nodes). Both continuous and range timelines are generated by monitoring a quantity called *summary-based variation*, which is defined for summaries during the course of topic evolvement. A large variation at a particular moment implies a sub-topic change, leading to the addition of a new node on the timeline.

The main contributions of our work include: (1) A continuous tweet stream summarization framework; (2) Novel data structures and algorithms for online summarization and historical summarization of any arbitrary time interval; (3) A topic evolvement detection scheme for continuous and range timelines; (4) Extensive experiments on real Twitter datasets, showing promising results in terms of summary quality and efficiency.

The rest of the 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*, *traditional document summarization* and *microblog summarization and mining*.

2.1 Stream Data Clustering

Stream data clustering has been widely studied in the literature. 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 is 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 [7][10][27][29]. 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 [2], the authors propose to generate duration-based clustering results by extending CluStream for text and categorical data stream. However, this algorithm relies on an online phase to generate large number of "micro-clusters", leading to inefficiency and poor storage utilization.

2.2 Traditional Document Summarization

Document summarization techniques can be categorized into two types: *extractive* techniques and *abstractive* ones. The former selects sentences from the documents, while the latter may generate phrases and sentences that do not ap-

pear 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 [3][30] [19][26] [6][22]. Some works try to extract sentences without such salient scores. A method using Singular Value Decomposition (SVD) is proposed in [8] to select highly ranked sentences. Wang et al. [23] use the Symmetric Nonnegative Matrix Factorization (SNMF) to cluster sentences and choose sentences in each cluster for summarization. In [11], He et al. propose to summarize documents from the perspective of data reconstruction, and select sentences that can best reconstruct the original documents. Unfortunately, all above methods neglect the significant temporal dimension of documents, which is critical in our problem.

Yan et al. propose a technique called Evolutionary Timeline Summarization (ETS) [24] to compute evolution timelines consisting of individual but correlated component summaries. However, the dates of component summaries are determined by a pre-defined timestamp set. In contrast, our solution discovers the changing dates and generate timelines dynamically during the process of continuous summarization. More importantly, ETS does not focus on efficiency and scalability. Its task is formulated as an optimization problem via iterative substitution. Thus it does not meet our requirements.

2.3 Microblog Summarization and Mining

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 multiple tweet posts on the same topic with a single tweet [20]. Later, Inouye et al. proposed a modified Hybrid TF-IDF algorithm and a Cluster-based algorithm to generate multiple post summaries [13]. In [9], Harabagiu et al. introduced a framework for microblog summarization which capitalizes on a combination of two relevance models: an event structure model and a user behavior model. Takamura et al. proposed a microblog summarization method based on the p-median problem, which takes posted time of microblogs into consideration [21].

The emergence of microblogs also motivates research on other mining tasks, including topic modeling [12], storyline generation [14] and event exploration [17]. Most of these researches focus on static datasets instead of data streams. Yang et al. studied twitter stream analysis [25], but they aim at frequent pattern mining and compression, which is also a different problem from ours.

To sum up, almost all existing document/microblog summarization works mainly deal with static and small datsets, and rarely pay attention to evolvement and efficiency issues.

3. PRELIMINARIES

In this section, we first give a data model for tweets. Then we introduce two data structures used in our solution, namely 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 having 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 [5].

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 $t_1, t_2, ..., t_n$, its **Tweet Cluster Vector (TCV)** 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 **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 **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 **cv**):

$$\mathbf{cv} = (\sum_{i=1}^{n} w_i \cdot \mathbf{tv_i})/n = \mathbf{wsum\_v}/n \quad (1)$$

The definition of TCV is an extension of the cluster feature vector in [28]. Like in [28], our TCV structure can also be updated in an incremental way when new tweets arrive. We shall discuss details on updates to TCV 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. *The **Pyramidal Time Frame (PTF)** stores snapshots at differing levels of granularity depending on the recency. Snapshots are classified into different* orders *which vary from 0 to $log(T)$, where T is the time elapsed since the beginning of the stream. The order of a particular class 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 i-th order occur at time intervals of α^i, where α is an integer and $\alpha \geq 1$. Specifically, each snapshot of the i-th 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.*

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

According to the definition, PTF has two properties: (1) The maximum order of any snapshot stored at T timestamps 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 stored 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.

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

In this section, we present the details of our Sumblr framework. As shown in Figure 2, our framework includes two main modules: the tweet stream clustering module and the high-level summarization module. In what follows, we will elaborate these two modules respectively.

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 statistics, with only one scan on the data.

4.1.1 Initialization

At the beginning of the stream, we collect a small number of tweets and use a k-means clustering algorithm to create the initial clusters. The value of k and the initial cluster centroids are decided via the Canopy [18] method. Once the initial clusters are established, the first set of TCVs are initialized according to Definition 1. Next, the stream clustering process starts to incrementally update the TCVs when 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. First, we try to absorb t into one of the current clusters. The priority is given to the cluster whose centroid is the closest to t. Specifically, we get the centroid (denoted as co) of a cluster based on Equation (1), compute the cosine similarity between co and t, and find the cluster C_p with $MaxSim(co, t)$.

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 will be created. The decision of whether to create a new cluster can be made with the following heuristic.

Algorithm 1: Incremental tweet stream clustering

Input: a cluster set S

```
while !stream.end() do
    Tweet t = stream.next();
    choose C_p in S whose centroid is the closest to t;
    if MaxSim(co_p, t) < MBS then
        create a new cluster C_new = {t};
        S = S ∪ C_new;
    else
        update C_p with t;
    if TS_current % (α^i) == 0 then
        store S into PTF;
```

HEURISTIC 1. *If $MaxSim(co, t)$ is smaller than a **Minimum Bounding Similarity** (MBS), then t is upgraded to a new cluster. Otherwise, t is added to its closest cluster.*

The MBS is defined as $\beta \cdot \overline{Sim(co, t_i)}$, where β is a bounding factor ($0 < \beta < 1$) and $\overline{Sim(co, t_i)}$ is the average cosine similarity between co and tweets included in C_p. According to Section 3.2, $\overline{Sim(co, t_i)}$ can be calculated by using the information stored in TCV:

$$\overline{Sim(co, t_i)} = \frac{1}{n}\sum_{i=1}^{n}\frac{\mathbf{tv_i} \cdot \mathbf{cv}}{||\mathbf{tv_i}|| \cdot ||\mathbf{cv}||} = \frac{\mathbf{cv}}{n \cdot ||\mathbf{cv}||}\sum_{i=1}^{n}\frac{\mathbf{tv_i}}{||\mathbf{tv_i}||}$$

$$= \frac{\frac{\mathbf{wsum\_v}}{n}}{n \cdot ||\frac{\mathbf{wsum\_v}}{n}||} \cdot \mathbf{sum\_v} = \frac{\mathbf{wsum\_v} \cdot \mathbf{sum\_v}}{n \cdot ||\mathbf{wsum\_v}||}$$

When adding 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 can not 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 gives an overview of our incremental clustering procedure.

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 m is

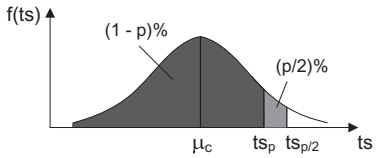


Figure 3: Probability density func. of timestamp

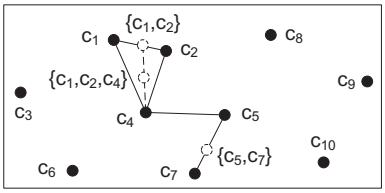


Figure 4: A running example of cluster merging

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 stream, timeliness is important because they usually do not last for a long time. Therefore it is safe to delete the tweet 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 requirements, especially when some 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 Figure 3). Then, we can approximate Avg_p using the $(100 - p/2)$-th percentile (noted as $ts_{p/2}$ in Figure 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

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

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* and can be calculated using the Maclaurin series expansion [31].

The value of $ts_{p/2}$ represents the *freshness* of cluster C. We empirically set a freshness threshold as 3 days (as empirically no bursty events would last longer) and set $p = 10$. If $ts_{p/2}$ is smaller than this threshold, i.e., the average timestamp of the latest 10 percent tweets is more than 3 days old, then we regard C as an outdated cluster and remove it.

4.1.4 Merging Clusters

If the number of clusters keeps increasing and few of them are deleted, the 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 until the termination condition is satisfied. First, we sort all cluster pairs by their centroid similarities in a descending order. Then, beginning with the most similar pair, we try to merge two clusters in the pair. 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), then 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 merge coefficient which provides a balance between available memory space and the quality of remaining clusters). We omit the pseudo-code of the algorithm due to space limitation.

During cluster merging, each composite cluster is given an $IDList$ which consists of IDs of the clusters merged in it. Furthermore, its TCV is obtained by the *Aggregation operation* to combine two TCVs.

DEFINITION 3. ***(Aggregation Operation)*** *Let C_1 and C_2 be two clusters, and their TCV structures be $TCV(C_1)$ and $TCV(C_2)$. Then, 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 newly merged centroid.*

Figure 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: $(c_1, c_2), (c_2, c_4), (c_1, c_4), (c_5, c_7), (c_4, c_5), ...$ 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 next pair, c_1 and c_4 both have been merged, so this pair is 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 $C^2_{N'} + 1 = N'(N'-1)/2 + 1$ pairs.

4.2 High-level Summarization

The high-level summarization module provides two types of summaries: *online* and *historical* summaries. The online summaries are retrieved directly from the current clusters

maintained in the memory. For historical summaries, retrieval of the required clusters is more complicated. In what follows, we shall focus on the second type.

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*

- $\mathbf{sum\_v = sum\_v_1 - sum\_v_2}$
- $\mathbf{wsum\_v = wsum\_v_1 - wsum\_v_2}$
- $ts1 = ts1_1 - ts1_2$
- $ts2 = ts2_1 - ts2_2$
- $n = n_1 - n_2$
- *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

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 $D(c)$. Our tweet summarization aims to extract k tweets from T, so that they can cover as many tweet contents as possible.

We first prove it is a NP-hard problem, then we present a greedy algorithm to solve it.

LEMMA 1. *The tweet summarization problem is NP-hard.*

PROOF. Let us first describe this problem formally. $\mathcal{F} = \{T_1, T_2, ..., T_t\}$ is 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. The subsets may have some tweets in common because one tweet can be related to more than one sub-topic. 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 notice that this is the *Max-k-Cover* problem, which is NP-hard. Therefore, our summarization problem is also NP-hard. □

More generally, summary length are limited in terms of words (250 words). Since the number of words and that of tweets are linearly dependent, the problem is still 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

```
S = ∅, T = {all the tweets in ft_sets of D(c)};
Build a similarity graph on T;
Compute LexRank scores LR;
T_c = {tweets with the highest score in each cluster};
while |S| < L do
    foreach tweet t_i in T_c do
        calculate v_i according to Equation (2);
    select t_max with the highest v_i;
    S = S ∪ t_max;
while |S| < L do
    foreach tweet t'_i in T − S do
        calculate v'_i according to Equation (2);
    select t'_max with the highest v'_i;
    S = S ∪ t'_max;
return S;
```

We design a greedy algorithm to select representative tweets to form summaries (Algorithm 2). First, we gather tweets from the ft_sets in $D(c)$ as a set T, and build a cosine similarity graph. 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 [6] to compute centrality scores for tweets. LexRank is an effective static summarization method and is efficient for small-sized datasets. But when datasets 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 according to:

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

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)* [4]. 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 ($t_i \in T - S$) according to Equation (2).

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 sec-

ond component can be updated incrementally. So the worst-case cost of tweet selection is $O(N^2 + (|S| - N) \cdot |T|)$. Since $|S| << |T|$, 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 Topic Evolvement Detection

Topic evolvment detection algorithm can produce continuous and range timelines in a similar way. We shall only describe the continuous case here.

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

$$D_{KL}(S_c||S_p) = \sum_{w \in V} p(w|S_c) ln \frac{p(w|S_c)}{p(w|S_p)}, \quad (3)$$

where

$$p(w|S_c) = \frac{tf(w, S_c)}{\sum_{w'} tf(w', S_c)}$$

$$p(w|S_p) = \frac{tf(w, S_p) + \epsilon}{\sum_{w'}(tf(w', S_p) + \epsilon)}$$

V is the vocabulary set and $tf(w, S)$ is the term frequency for word w in S. ϵ is a small positive constant for Laplace Smoothing, which is applied to avoid zero values of $p(w|S_p)$. Furthermore, we normalize the distance into interval $[0, 1)$ using $D = 1 - e^{-D_{KL}}$, for ease of comparison.

According to the summary-based variation, we determine if the current time is a sub-topic changing node by:

$$\frac{D_{cur}}{D_{avg}} > \tau,$$

where D_{cur} is the distance between 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 ($\tau \geq 1$) is the decision threshold.

That is, we detect topic evolvement when there is a *burst* in distances between successive summaries. In this way, we can automatically draw a timeline as the stream proceeds.

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.

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.

Table 2: Basic information of 5 datasets

Topics (filtering keywords)	#Tweets	Time Span
Obama	95,055	2009.2 - 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

5. EXPERIMENTS

5.1 Experimental Setup

Datasets We construct 5 datasets to evaluate Sumblr. One is obtained by conducting keyword filtering on a Twitter dataset (from Feb. to Oct., 2009) used by [5]. The other four include more recent tweets acquired in one month via Twitter's keyword tracking API [2]. 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 datasets 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 huge size of the datasets.

Thus, we employ a two-step method to obtain fair-quality reference summaries without tremendous human labor:

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 [13]: first clusters the tweets and then summarizes each cluster by picking the most weighted post according to the hybrid TF-IDF weighting described in [13].

LexRank [6]: first builds a sentence similarity graph, and then selects important sentences based on the concept of eigenvector centrality.

DSDR [11]: models the relationship among sentences using linear reconstruction, and finds an optimal set of representative sentences to approximate the original documents, by minimizing the reconstruction error.

2) Next, for each subset, the final reference summary is manually extracted from three candidate summaries. Priority is given to those sentences appearing in at least two candidate summaries. The length of each summary is limited to 250 words.

Ground truth for timelines For the timeline generation test, the reference timelines are manually produced. We choose the "Arsenal" and "Chelsea" datasets for this experiment, as the ground truth for sport topics are relatively easier to build manually. Specifically, we read through all the related news during one month from news websites (Yahoo! and ESPN), and select those dates as nodes on the reference timeline when important events happen, e.g., football matches, players' signing of new contracts, etc. [3]

Baseline methods Most existing summarization methods are not designed to handle continuous summarization. However, they can be adapted to streaming data by using a *sliding window* scheme. As illustrated in Figure 8, 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.

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

[3]Our datasets and ground truth are available at http://db.zju.edu.cn/s/sumblr/.

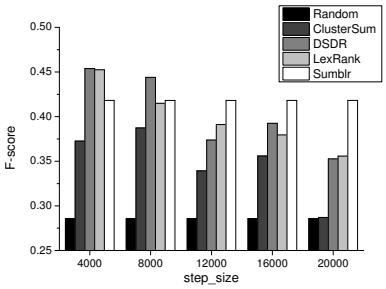


Figure 5: Quality on step size

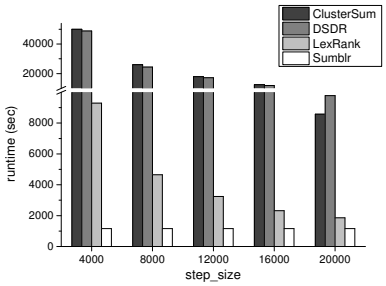


Figure 6: Efficiency on step size

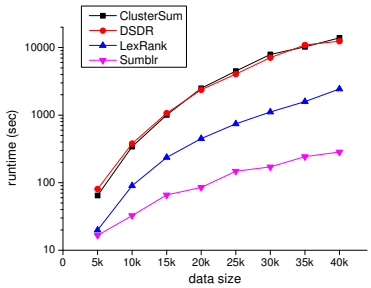


Figure 7: Scalability on data size

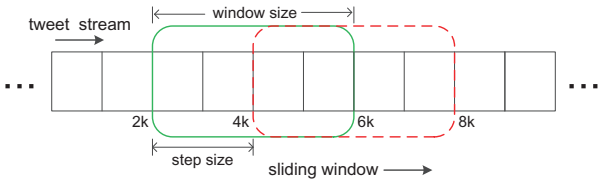


Figure 8: The sliding window mechanism

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 [15] for evaluation. Among supported metrics, ROUGE-1 has been demonstrated to be the most consistent with human judgement [16]. Considering the short and informal nature of the tweet contents, we decide that ROUGE-1 is suitable for measuring tweet summaries.

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

$$\int_{T_0}^{T} metric(t)dt \tag{4}$$

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 datasets.

5.2 Overall Performance Comparison

In this section, we compare the F-scores and runtime cost between Sumblr and the 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 20000, sampling interval to 2000. The step size varies from 4000 to 20000. The metrics are averaged over the whole stream.

Figure 5 and Figure 6 present the results for different step sizes. In Figure 5, 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 (4000), DSDR and LexRank achieve better summary quality than Sumblr, at the expense of much more computation. When $step\_size \geq 12000$, Sumblr outperforms all the baseline methods in terms of both summarization quality and computation cost. Although the efficiency of LexRank is comparable with our method when $step\_size \geq 16000$, its summary quality is significantly worse.

The above results reveal a major problem with the baseline methods: These methods rely on a small step size to produce quality summaries. Unfortunately, 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 here is that, as the ground truth is generated using these baseline methods, the summary quality is to some extent biased in favor of them.

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.

Figure 7 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 15000, 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 deletions and merges.

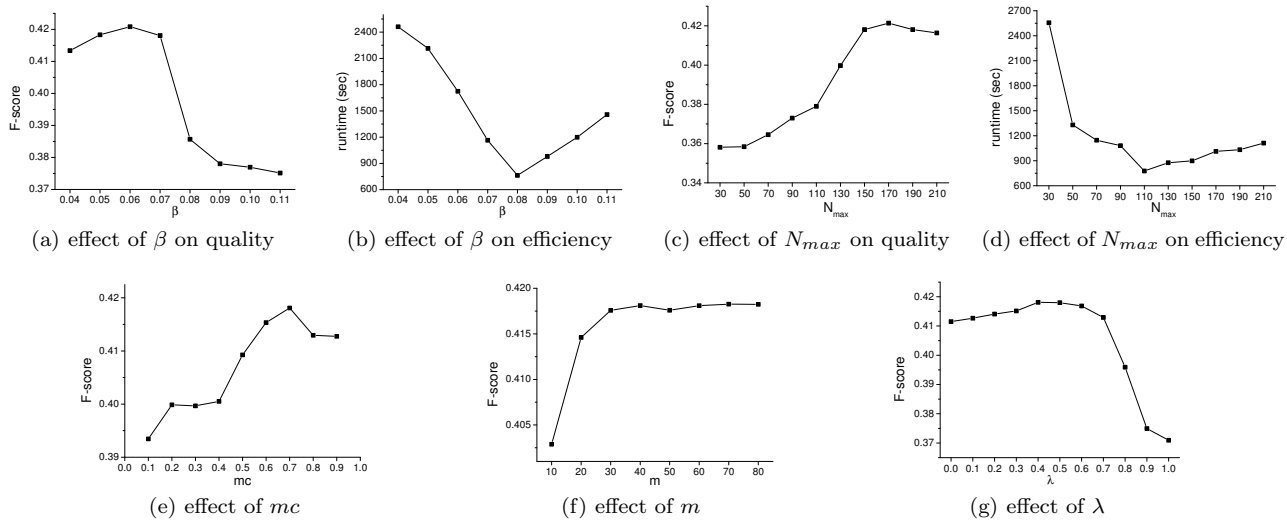
5.3 Parameter Tuning

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. Figure 9(a) and Figure 9(b) 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}. Figure 9(c) and Figure 9(d) depicts 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



(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

Figure 9: Performance of different parameters

efficiency, so we only present its quality results (Figure 9(e)). 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 Figure 9(f), 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 trade-off 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 Figure 9(g). When $\lambda \geq 0.7$, the extreme emphasis on coverage causes performance loss. Therefore, we set $\lambda = 0.4$ as a balanced factor.

5.4 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 dataset as the input stream. The interval between two successive snapshots (timestamp unit) is one hour. For a timestamp ts, we evaluate the results for drill-down/roll-up summaries using durations with different length len. Due to space limit, we only present the average F-score for len varying from 1 day to 10 days, and report $score(ts)$ by interval of 48 hours.

$$score(ts) = \frac{\sum_{len} F\text{-}score([ts - len, ts])}{10} \quad (5)$$

Figure 10 gives the following observations:

- There exists a common trend for all ls: as a time duration is closer to the current time, the summary quality improves. This is because PTF has finer granularity of snapshots for more recent moments. Thus the queried durations can be better approximated.
- For different values of l, the summary quality is similar for recent durations, but decreases in different degrees for early durations. The reason is that for recent moments, most of their snapshots or neighborhood snapshots are still kept in PTFs regardless of l; while for early moments, their snapshots are more likely to be removed from PTFs with smaller ls, due to smaller capacity of each order.

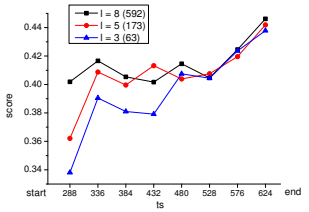


Figure 10: Quality on time duration

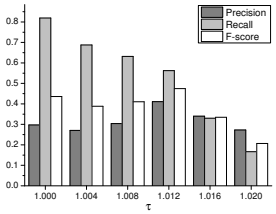


Figure 11: Effect of τ

- A larger l leads to better results but more storage cost (the numbers in the parentheses represent the amounts of snapshots in PTF). This is obvious, as a larger l enables PTF to store more snapshots, which results in more accurate approximation and 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.

5.5 Timeline Generation

In this section, we evaluate the effectiveness of topic evolvement detection. We present the precision, recall, and F-score of the timeline nodes detected by our algorithm while varying the decision threshold τ.

From Figure 11, we can see that the recall declines as τ increases. This is expected as higher threshold would exclude more promising candidates i.e., produce more false negatives. The precision and F-score both reach the highest value when $\tau = 1.012$. Although the recall drops when τ increases, precision increases further (as more false positive nodes are excluded). Surprisingly, when $\tau > 1.012$, precision also drops. This may result from the incompleteness of our manual timeline or noises in the datasets.

The output for "Arsenal" is presented in Table 3. It is interesting to find that our summary-based timeline not only detects important events (e.g. a match is called off due to a tube strike), but also contains popular public opinions (e.g. public calling for WENGER OUT).

6. CONCLUSION

We proposed a prototype called Sumblr which supported continuous tweet stream summarization. Sumblr employed

Table 3: Selected part of the timeline for "Arsenal"

12.12: 1. Yes Bradford! WENGER OUT. 2. Arsenal lose to Bradford: Should Arsene Wenger go? 3. Arsenal Chief Executive Ivan Gazidis has apologised to the club's fans, and described Arsenal's defeat to Bradford.	**12.19**: 1. Good news for Arsenal fans: Wilshere, Oxlade-Chamberlain, Ramsey, Gibbs and Jenkinson sign new contracts. 2. ♯afc Arsenal's Premier League match against West Ham on Boxing Day is called off because of a possible tube strike. 3. Five Arsenal players sign new contracts, but not one of those five is Theo Walcott. 4. Wenger: "We hope we will be capable of building a team around the young English players." ♯Arsenal ♯AFC
12.16: 1. Every arsenal fan knows how chelsea fans feel like right now. 2. Game over, we lost.. Am I sad? No are arsenal fans happy? Yes, are they gonna ever play world club cup? No. 3. RVP: "I'm sorry to Arsenal fans, I've never been happier than this. Finally I found a home and peace here. I'm happy at Man United."	**12.20**: 1. Champions League draw: Manchester United v Real Madrid, Arsenal v Bayern Munich, Celtic v Juventus. 2. So it is @Arsenal Vs Bayern Munich....good game we'll have :-)
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. 3. I know it's "only" Reading but still good to see Arsenal playing so well.	**12.22**: 1. Arsenal better win the game against wigan. 2. Wilshere hails Arsenal's fighting spirit after Wigan win.

a tweet stream clustering algorithm to compress tweets into TCVs and maintain them in an online fashion. Then, it used a TCV-Rank summarization algorithm for generating online summaries and historical summaries with arbitrary time durations. The topic evolvement could be detected automatically, allowing Sumblr to produce dynamic timelines for tweet streams. The experimental results demonstrated 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 datasets.

Acknowledgments

The work is supported by the National Science Foundation of China (GrantNo. 61170034).

7. REFERENCES

[1] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu. A framework for clustering evolving data streams. In *VLDB*, pages 81–92, 2003.

[2] C. C. Aggarwal and P. S. Yu. On clustering massive text and categorical data streams. *Knowl. Inf. Syst.*, 24(2):171–196, 2010.

[3] R. Barzilay and M. Elhadad. Using lexical chains for text summarization. In *Proceedings of the ACL Workshop on Intelligent Scalable Text Summarization*, pages 10–17, 1997.

[4] J. Carbonell and J. Goldstein. The use of mmr, diversity-based reranking for reordering documents and producing summaries. In *SIGIR*, pages 335–336, 1998.

[5] C. Chen, F. Li, B. C. Ooi, and S. Wu. Ti: an efficient indexing mechanism for real-time search on tweets. In *SIGMOD*, pages 649–660, 2011.

[6] G. Erkan and D. R. Radev. Lexrank: graph-based lexical centrality as salience in text summarization. *J. Artif. Int. Res.*, 22(1):457–479, 2004.

[7] L. Gong, J. Zeng, and S. Zhang. Text stream clustering algorithm based on adaptive feature selection. *Expert Syst. Appl.*, 38(3):1393–1399, 2011.

[8] Y. Gong and X. Liu. Generic text summarization using relevance measure and latent semantic analysis. In *SIGIR*, pages 19–25, 2001.

[9] S. M. Harabagiu and A. Hickl. Relevance modeling for microblog summarization. In *ICWSM*, 2011.

[10] Q. He, K. Chang, E.-P. Lim, and J. Zhang. Bursty feature representation for clustering text streams. In *SDM*, 2007.

[11] Z. He, C. Chen, J. Bu, C. Wang, L. Zhang, D. Cai, and X. He. Document summarization based on data reconstruction. In *AAAI*, 2012.

[12] L. Hong, A. Ahmed, S. Gurumurthy, A. J. Smola, and K. Tsioutsiouliklis. Discovering geographical topics in the twitter stream. In *WWW*, pages 769–778, 2012.

[13] D. Inouye and J. K. Kalita. Comparing twitter summarization algorithms for multiple post summaries. In *SocialCom*, pages 298–306, 2011.

[14] C. Lin, C. Lin, J. Li, D. Wang, Y. Chen, and T. Li. Generating event storylines from microblogs. In *CIKM*, pages 175–184, 2012.

[15] C.-Y. Lin. Rouge: A package for automatic evaluation of summaries. In *Proc. ACL workshop on Text Summarization Branches Out*, pages 74–81, 2004.

[16] C.-Y. Lin and E. Hovy. Automatic evaluation of summaries using n-gram co-occurrence statistics. In *HLT-NAACL*, pages 71–78, 2003.

[17] 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 *CHI*, pages 227–236, 2011.

[18] A. McCallum, K. Nigam, and L. H. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *KDD*, pages 169–178, 2000.

[19] D. R. Radev, H. Jing, M. Styś, and D. Tam. Centroid-based summarization of multiple documents. *Inf. Process. Manage.*, 40(6):919–938, 2004.

[20] B. Sharifi, M.-A. Hutton, and J. Kalita. Summarizing microblogs automatically. In *HLT-NAACL*, pages 685–688, 2010.

[21] H. Takamura, H. Yokono, and M. Okumura. Summarizing a document stream. In *ECIR*, 2011.

[22] X. Wan and J. Yang. Multi-document summarization using cluster-based link analysis. In *SIGIR*, pages 299–306, 2008.

[23] D. Wang, T. Li, S. Zhu, and C. Ding. Multi-document summarization via sentence-level semantic analysis and symmetric matrix factorization. In *SIGIR*, pages 307–314, 2008.

[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 *SIGIR*, pages 745–754, 2011.

[25] X. Yang, A. Ghoting, Y. Ruan, and S. Parthasarathy. A framework for summarizing and analyzing twitter feeds. In *KDD*, pages 370–378, 2012.

[26] W.-t. Yih, J. Goodman, L. Vanderwende, and H. Suzuki. Multi-document summarization by maximizing informative content-words. In *IJCAI*, pages 1776–1782, 2007.

[27] J. Zhang, Z. Ghahramani, and Y. Yang. A probabilistic model for online document clustering with application to novelty detection. In *NIPS*, 2005.

[28] T. Zhang, R. Ramakrishnan, and M. Livny. Birch: an efficient data clustering method for very large databases. In *SIGMOD*, pages 103–114, 1996.

[29] S. Zhong. Efficient streaming text clustering. *Neural Networks*, 18(5-6):790–798, 2005.

[30] Q. Zhou, L. Sun, and J. Nie. Is_sum: A multi-document summarizer based on document index graphic and lexical chains. *DUC2005*, 2005.

[31] D. Zwillinger. *CRC standard mathematical tables and formulae*. CRC press, 2011.