Adjustable Time-Window-Based Event Detection on Twitter

Qinyi Wang[1], Jieying She[2], Tianshu Song[1], Yongxin Tong[1(✉)], Lei Chen[2], and Ke Xu[1]

[1] SKLSDE Lab and IRC, Beihang University, Beijing, China
{wangqinyi,songts,yxtong,kexu}@buaa.edu.cn
[2] The Hong Kong University of Science and Technology, Hong Kong SAR, China
{jshe,leichen}@cse.ust.hk

Abstract. Twitter has become an important platform for reporting breaking news and instant events. However, it is almost impossible to detect events on Twitter manually due to the large volume of data and the noise in them. Though automatic event detection has been studied a lot, most works can only detect events in a fixed time window. In this paper, we propose an efficient system that can detect events in adjustable time windows. We detect terms with unusual frequency and group them into events. We further modify a segment tree data structure to support adjustable time window based event detection, which can efficiently aggregate statistics of terms of varied-sized time windows and is both space and time saving. We finally validate the effectiveness and efficiency of our proposed techniques through extensive experiments on real datasets.

1 Introduction

Twitter has become an important platform for fast reporting and broadcasting of news and events these days mainly due to it convenience and speed of spreading information. However, due to the large volume of data generated, monitoring and detecting events manually is infeasible and automatic detection is required. Although automatic event detection has been well studied in traditional media, i.e. the Topic Detection and Tracking (TDT) research program, event detection on Twitter is much more challenging due to its noisy data.

Some event detection techniques have been proposed recently. However, they fail to meet the following adjustable time window based event detection requirement. Suppose some users want to know what is happening around the world by monitoring Twitter in real-time. They would like to be reported breaking news or the hottest events during the past 120 min. When spotting a hot event in the past 120 min, they want to learn about how it developed and thus would like to check what happened during the past 60, 30 and 10 min.

Existing techniques cannot solve the above scenario due to the following reasons. First, they do not support detecting events in adjustable sizes of time windows and only divide the timeline into fixed time windows where events are

B. Cui et al. (Eds.): WAIM 2016, Part II, LNCS 9659, pp. 265–278, 2016.
DOI: 10.1007/978-3-319-39958-4_21

to be detected, which may not adapt to detecting different types of events or attract users desiring various degrees of real-time. Second, many works only focus on detecting category-specific or topic-specific events [8,12], which cannot attract users with broad interests. Third, some techniques, e.g. learning topic models [11], were too complex to support real-time application, which is important in case of breaking news and emergent events. Finally, some unsupervised techniques required several parameters or thresholds to detect events by identifying bursty terms [8], which could be sensitive to the parameter settings and hard to adapt to the dynamic environment of Twitter.

In this paper, we address the above problems with a solution with the following features. First, our solution supports real-time applications. We perform simple operations on the data with low latency, enabling timely event report. The time and memory consumed to detect events should not increase as time elapses and only recent statistics of data will be kept while obsolete ones will be discarded. Second, our solutions is completely unsupervised and no labeled data is required. Third, our solution is free of thresholds to identify bursty terms. Finally, we design a space-saving data structure that can efficiently detect events w.r.t. adjustable sizes of time windows.

Our approach achieves the goals by detecting unusually frequent words within an adjustable time window. We assume that an emerging event is indicated by a group of suddenly frequent terms, and we define a burst score for each term and group bursty terms based on their co-occurrence into candidate events. To enable adjustable time window based event detection, we modify the segment tree, Streaming Timeline Tree (ST-tree). More specifically, we first assume a minimum time window as a time unit, and an adjustable time window as several continuous time units. By properly storing statistics that we need to calculate scores for terms in the nodes and manage obsolete nodes and new ones, we are able to support adjustable time window based event detection.

The major contributions of our paper include:

- We propose a simple, effective and efficient method to detect open-domain emerging events.
- We design a data structure to support adjustable time window based event detection.
- We verify the effectiveness and efficiency of the proposed methods through extensive experiments on real datasets.

The rest of the paper is organized as follows. In Sect. 2, we formally define our problem. We then describe in detail our event detection algorithm in the Sect. 3. In Sect. 4, we present our ST-tree for flexible time frame event detection. Evaluation of our proposed methods is conducted in Sect. 5. We review previous works in Sect. 6 and conclude this work in Sect. 7.

2 Problem Formulation

We address our problem in a real-time environment, where we continuously receive and process streaming data from Twitter. A group of consecutive time

windows (time frame) $f_{c-n+1}, \cdots, f_c$ are some non-overlapping time intervals of equal size on the timeline, where f_c is the most recent one. A time window is the basic unit which we identify events for. Note that "time frame" and "time window" are the same meaning and they are used interchangeably in this paper.

The major characteristic of an emerging event is that more and more people start to talk about it and some relevant terms, which will exhibit unusually higher frequency patterns than past records. We name them as abnormal terms. Three factors determine whether a term is abnormal or not: term frequency, the number of users using the term (i.e. user frequency), and its frequency statistics in previous time windows. More details will be described in the next section. In our work, we detect events with a sliding time window setting.

Definition 1 (Sliding time Window). *Given the current time window f_c with length L starting at time t_i and ending at $t_i + L$, the next sliding time window f_{c+1} begins at time $t_i + G$ and ends at $t_i + G + L$, where G is the sliding gap such that $G < L$ and L is divisible by G.*

Definition 2 (Event Detection). *The event detection problem is to identify groups of abnormal terms within the current time window and report the most abnormal groups as possible events in a sliding time window setting.*

We next define our adjustable time window based event detection problem.

Definition 3 (Time Unit). *A time unit is the minimal size of the time window available.*

Definition 4 (Adjustable Time Window). *An adjustable time window is one that consists of several consecutive time units, and the number of time units is specified by users.*

We limit the maximum number of time units an adjustable time window can consist. The reason is that a very long time window is uninteresting to Twitter users as mainstream media will catch up and report the events later. Though a larger time window may be applicable to event tracking or summarizing, we focus on detecting emerging events and thus define a maximum time window.

Definition 5 (Adjustable Time Window Based Event Detection). *The adjustable time window based event detection problem is to identify groups of abnormal terms within the current adjustable time window and report the most abnormal groups as possible events in a sliding time window setting with a sliding gap equal to the length of a time unit.*

3 Event Detection

3.1 System Overview

Our event detection system consists of two stages of processing. The first stage is to process each new tweet and store some statistics, and the second stage is to identify events at the end of the current time window.

3.2 Processing of Tweets

For each new tweet, we use unigrams as terms, since we find that unigrams outperforms n-grams in both effectiveness and efficiency. Details will be presented in Sect. 5. We then update the statistics of the current time window f_c: the number of tweets cd_{f_c} within f_c and for each term w a set of ids $tweet\_set_{f_c,w}$ referring to the tweets that contain the term and its frequency $cw_{f_c,w}$. For each term, we further maintain $uw_{f_c,w,u}$ for each user u, which is the number of tweets published by u during f_c that contain w, and $uwt_{f_c,w}$:

$$uwt_{f_c,w} = \sum_u (2 - \frac{1}{2^{uw_{f_c,w,u}-1}})(uw_{f_c,w,u} > 0) \tag{1}$$

We observe that around 30 % of tweets are retweets, many of which are generated by followers of celebrities, which makes the corresponding cw and uwt extremely large and thus results in false alarms. Note that breaking news could also trigger a large number of retweets, but it has longer retweeting chains and more origins (i.e. the original authors). Thus, when dealing with a tweet published by u_r and retweeted from u_o, we update uw_{f_c,w,u_o} instead of uw_{f_c,w,u_r} to reduce false alarms caused by retweets.

3.3 Identification of Abnormal Terms

Identification of abnormal terms is performed at the end of the current time window. We define significance score $ss_{f_c,w}$, abnormality score $as_{f_c,w}$ and abnormal terms as follows. Note that when calculating the average scores $ss_{avg,w}$ over past time windows, we use the most recent past non-overlapping time windows.

$$ss_{f_c,w} = \frac{cw_{f_c,w}}{cd_{f_c}} * \frac{uwt_{f_c,w}}{cd_{f_c}} \tag{2}$$

$$ss_{avg,w} = \frac{cw_{avg,w}}{cd_{avg}} * \frac{uwt_{avg,w}}{cd_{avg}} \tag{3}$$

$$as_{f_c,w} = \frac{ss_{f_c,w}}{ss_{avg,w}} \tag{4}$$

Definition 6 (Abnormal Term). *A term with abnormality score larger than 1 is abnormal.*

Notice that we simply regard all terms with scores higher than historical records as abnormal, as any threshold is sensitive to the fast changing data and it is impossible to define a universal threshold to effectively select terms under our unsupervised scheme.

3.4 Clustering Abnormal Terms

We then group terms into events. The main idea is to cluster two terms based on how often they co-occur as words that co-occur often are more likely to refer to the same event. We introduce a threshold *cluster_thres* to determine whether two terms should be grouped together, which controls how cohesive a cluster is.

We maintain a set $term\_set_{f_c,m}$ for each tweet m containing abnormal terms, which consists of all the abnormal terms mentioned in m. Such sets can be easily created from *tweet_set* of abnormal terms after we identify them. And we have the following lemma stating under what condition two terms co-occur.

Lemma 1. *Given a term w_i and $tweet\_set_{f_c,w_i}$, any term $w_j(\neq w_i) \in \cup_{m_i \in tweet\_set_{f_c,w_i}} term\_set_{f_c,m_i}$ must co-occur with w_i for at least once. Any other term $w_k(\neq w_i) \notin \cup_{m_i \in tweet\_set_{f_c,w_i}} term\_set_{f_c,m_i}$ does not co-occur with w_i.*

Then during clustering, we find highly co-occurring terms for each term w_i by scanning each term in $\cup_{m_i \in tweet\_set_{f_c,w_i}} term\_set_{f_c,m_i}$ and counting the frequency of the terms. The process is illustrated in Algorithm 1.

Algorithm 1. Clustering Abnormal Terms

```
   input : Abnormal terms {w_1, ..., w_n}, {tweet_set_{f_c,w_i}}, {term_set_{f_c,m_i}}
   output: Clusters {C_1, C_2, ..., C_m}
1  Mark each term as one cluster containing only itself;
2  foreach i ← 1 to n − 1 do
3      foreach m_i ∈ tweet_set_{f_c,w_i}, w_j ∈ term_set_{f_c,m_i} do
4          if w_i and w_j are already in the same cluster then
5              continue;
6          if w_j ∉ candidate then
7              Add w_j to candidate, cnt_{w_j} ← 1;
8          else
9              cnt_{w_j} ← cnt_{w_j} + 1;
10     foreach w_j ∈ candidate do
11         if cnt_{w_j} / min(|tweet_set_{f_c,w_i}|, |tweet_set_{f_c,w_j}|) > cluster_thres then
12             Combine the clusters containing w_i and w_j;
13 return Clusters {C_1, C_2, ..., C_m}
```

3.5 Ranking Events

Finally, we return the top-k ranked clusters based on their abnormal terms:

Definition 7 (Rank of Clusters). *Given two clusters C_i and C_j, C_i ranks higher than C_j if and only if $as_{f_c,w_{i_1}} > as_{f_c,w_{j_1}}$, or $as_{f_c,w_{i_1}} = as_{f_c,w_{j_1}}$ and $as_{f_c,w_{i_2}} > as_{f_c,w_{j_2}}$, where w_{i_1}, w_{i_2} are the two terms of C_i with the highest and second-highest abnormality scores respectively, and w_{j_1}, w_{j_2} are the two of C_j with the highest and second-highest abnormality scores respectively.*

3.6 Complexity Analysis

The major time-consuming component is clustering, whose complexity is $O(n * M * K)$, where n is the number of abnormal terms, M is the maximum size of a *tweet_set*, and K is the maximum size of a *term_set* (relatively small). We maintain six statistics for the current time window, and three for the past few time windows, whose number is $V * L/G$, where V is the number of past time windows. Thus, the total space complexity is $O(2W(VL/G+1)+UW+2MW)$, where W is the number of terms, and U is the maximum number of users.

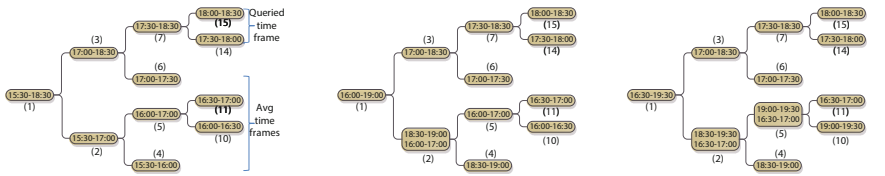
4 Adjustable Time Window Event Detection

In this section, we introduce the ST-tree structure. Let U_L denote the length of a time unit, then $L_M * U_L$ is the maximum length of time window supported.

4.1 Description of ST-tree

Basically, ST-tree is almost identical to segment tree that allows efficient query for statistics of an arbitrary interval. We update ST-tree in a different way. We first present some basic concepts of ST-tree in this subsection.

ST-tree is a binary tree. Each leaf represents a time unit, and each parent is the union of the time intervals its children refer to. We have $(V + 1) * L_M$ time units(leaves) in ST-tree, and where V is the number of time windows we average on. Suppose the current time unit is tu_c, the root then covers the whole time interval starting from the starting point of unit $tu_c - (V + 1) * L_M + 1$, and ending at the ending point of tu_c. Figure 1a shows an example of ST-tree.



(a) The Original ST-tree (b) After the 1st Update (c) After the 2nd Update

Fig. 1. An example of ST-tree with time units of length 30 min ($L = V = 2$).

Algorithm 2. Build ST-tree

input: Current node x, st and ed
Mark x as representing $[st, ed)$;
if *Length of* $[st, ed) > U_L$ **then**
  Create children x_{child_left} and x_{child_right} of x;
  $middle \leftarrow st + \lfloor \frac{(ed-st)}{2U_L} \rfloor U_L$;
  Run 2 with input x_{child_left}, st and $middle$;
  Run 2 with input x_{child_right}, $middle$ and ed;

Algorithm 2 presents how the structure of ST-tree is built during initialization, starting from the root given the initial interval $[st, ed]$.

Each node of ST-tree stores three statistics for the time interval it represents: cw, uwt and cd. When processing the current time unit, we could directly update the three statistics stored at the responding leaf by creating index for the leaves and visit them directly as we process the leaves from left to right in order. Updates for the other nodes are initiated when we arrive at the end of the current time unit. The detailed procedure will be described soon.

Besides the ST-tree, we also store *tweet_set* in the most recent L_M time units. Merging of the tweet sets of the most recent L time units for each term will be performed given a query for events in time window of length $L{*}U_L (1 \leq L \leq L_M)$.

4.2 Update

4.2.1 Calculation of Statistics

When at the end of the current time unit, we update ancestors of the current leaf. The statistics stored at a parent representing time interval f_{parent} are following (5) to (7), where f_{child_left} and f_{child_right} are the time intervals represented by its children.

$$cw_{f_{parent},w} = cw_{f_{child\_left},w} + cw_{f_{child\_right},w} \tag{5}$$

$$uwt_{f_{parent},w} = uwt_{f_{child\_left},w} + uwt_{f_{child\_right},w} \tag{6}$$

$$cd_{f_{parent}} = cd_{f_{child\_left}} + cd_{f_{child\_right}} \tag{7}$$

The true value of $uwt_{f_{parent},w}$ should not be the sum of $uwt_{f_{child\_left},w}$ and $uwt_{f_{child\_right},w}$ following the original definition of uwt. However, since storing a user set for each term in each node is too space-consuming, we instead store a single value and calculate the value of uwt by (6). In this way, we reduce our space and time consumption by relaxing the constraints on term's user frequency.

The whole process of updating is as follows: starting from the parent of the current leaf, we visit each node in the path from the current leaf to the root and update the corresponding statistics following (5) to (7).

4.2.2 Dealing with Obsolete Nodes

When processing the first $(V + 1)L_M$ time units, all we have to do is to fill in the values of the nodes. However, starting from the $(V + 1)L_M + 1$ time unit, we should keep the most recent statistics and discard the obsolete ones, as the statistics of unit $tu_c - (V + 1)L_M + 1$ will no longer be used when we proceed to the next unit $tu_c + 1$. We also need to find a node to store statistics of $tu_c + 1$.

Removing or adding nodes to the original tree will destroy its balanced structure. Thus, we propose to use the leaf representing $tu_c - (V + 1)L_M + 1$ to store statistics of $tu_c + 1$, so the oldest leaf becomes the newest leaf after update. Figures 1b and c show how the ST-tree changes after two updates of the original one in Fig. 1a.

Algorithm 3. Update of ST-tree

input : ST and the current leaf x
output: Updated ST
1 Mark each term as one cluster containing only itself;
2 **while** *x is not the root of ST* **do**
3 $x \leftarrow x$'s parent;
4 Update interval and statistics of x following (5) to (7);
5 **return** ST

Algorithm 4. Return Statistics of a Given Time Interval

input : Current node x, ST and $[t_s, t_t)$
output: $\{cw_{[t_s,t_t),w_i}\}, \{uwt_{[t_s,t_t),w_i}\}$ and $cd_{[t_s,t_t)}$
1 Initialize $\{now\_cw_{w_i}\}, \{now\_uwt_{w_i}\}$ and now_cd to 0;
2 **if** *The interval of $x \subset [t_s, t_t)$* **then**
3 **return** $\{cw_{w_i}\}, \{uwt_{w_i}\}$ *and cd stored at x*
4 **else**
5 **foreach** *Each child x_c of x* **do**
6 **if** *The interval of x_c overlaps with $[t_s, t_t)$* **then**
7 Run Algorithm 4 with input x_c, ST and $[t_s, t_t)$, aggregate the returned results to $\{now\_cw_{w_i}\}, \{now\_uwt_{w_i}\}$ and now_cd;
8 **return** $\{now\_cw_{w_i}\}, \{now\_uwt_{w_i}\}$ *and* now_cd

The process is illustrated in Algorithm 3. We use the oldest leaf to store statistics for the coming unit, and update its time interval. When arriving at the end of the newest time unit, we update its ancestors as usual. The time interval represented by a node may no longer be continuous.

4.3 Query

The query process is identical to that of our original problem, except that we first need to obtain the statistics for the queried time window and the past V time window from ST-tree, which is equivalent to the following problem: given a time interval $[t_s, t_t)$, return its statistics. The process is performed by running Algorithm 4 with input root of ST, ST, and $[t_s, t_t)$.

Note that Algorithm 4 applies to queries issued in the first $(V + 1)L_M$ time units as well as those issued afterwards. For queries issued after the first $(V + 1)L_M$ time units, the time intervals represented by the nodes may not be continuous. However, in such case, line 3 of Algorithm 4 will not be executed and we will keep exploring the children of such nodes until we reach at a node representing a continuous time interval that is subset of $[t_s, t_t)$.

Example 1. Suppose the current ST-tree is Fig. 1c, and we query for the statistics of 17:30-19:00. By running Algorithm 4, we will visit nodes 1, 2, 4, 3 and 7 in order, and aggregate the statistics stored in node 4 and 7.

4.4 Complexity Analysis

We compare ST-tree with two naive methods. One is Space-Severe (SS), which stores statistics for all possibly used time windows. Another one is Time-Severe (TS), which stores statistics only for time units and aggregates the results during querying. We only count the number of time windows/units/nodes.

Let $N = L_M * (V + 1)$ denote the number of leaves in ST-tree. The space complexity of ST-tree is $O(N)$. SS takes $O(NL_M)$ space, and TS takes $O(N)$ space. ST-tree takes $O(N)$ to build the tree and $O(logN)$ time to update. SS takes $O(L_M)$ time to update, and TS takes only $O(1)$ time. For query, the averaged time taken by ST-tree is $O(logN)$. SS takes $O(V)$ time, and TS takes $O((V + 1)L)$ time.

5 Experimental Study

5.1 Experimental Setup

We perform our experiments on machines with 2xIntel E5-2650 (8-core, 2 GHz, 20 M cache) CPU and 64G DDR3-1333 RAM. For all the experiments, we select the top-20 clusters detected in each time window and output the top-20 terms with the highest abnormality scores.

Twitter. We collected in total 11,625,484 tweets through Twitter's streaming API[1] continuously from Oct 31 to Nov 6, 2013. Only tweets written in English are used for evaluation. We use the data on October 31 to calculate past statistics, and detect events since Nov 1. The statistics are presented in Table 1.

Wikipedia. We refer to Wikipedia's current eventsportal, which keeps a list of daily events around the world with description of a few sentences, to see whether our algorithm can identify such events.

The Wikipedia data is processed as follows. For each event, we first extract non-stop words from its description as keywords. Second, we select 48 events that are ever mentioned in the Twitter dataset. Finally, we follow the links provided by Wikipedia to the background articles to check the exact time when mainstream media reported the events if available. When evaluating our algorithm, we do not count any cluster reported with delay of more than one day. To check whether a certain event is detected, we select clusters that contain at least two keywords of the event and then check manually whether the selected clusters contain highly relevant keywords of the events.

Table 1. Statistics of Collected Data

Number of retweets	3,488,124
Number of tweets per minute (on average)	1,153
Number of users	6,418,278
Number of different retweeted users	1,268,762

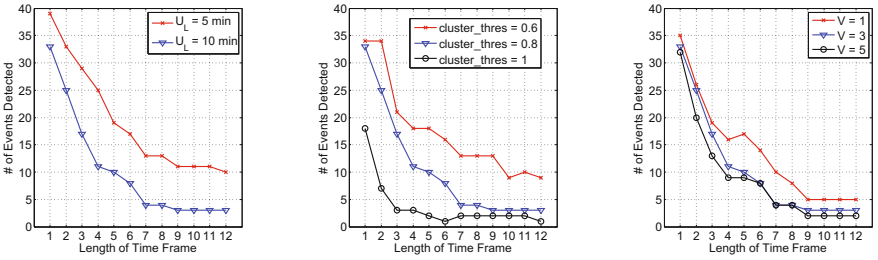
[1] https://dev.twitter.com/docs/api/1.1/get/statuses/sample.

5.2 Effectiveness Evaluation

We next evaluate our effectiveness in both aspects: how many events are detected and the impact of different parameter settings.

Accuracy. We compare with TwiCal[11] and Twevent [7] in this part of evaluation. TwiCal maintains a continuously updating demo[2], and thus we can directly obtain a list of events detected by TwiCal. Since we follow the methods and settings of Twevent to detect events, we also set $V = 3$, $U_L = 10$ min and $clu\_thres = 0.8$ for our proposed algorithm.

We present the results in Table 2. The only event that is detected by TwiCal but not by the others is that India launches a PSLV-XL rocket. The results indicate that our algorithm detects more events than both TwiCal and Twevent do. One possible reason for the bad performance of TwiCal is that it uses supervised technique and annotates only a very small number of tweets to extract event phrases, and thus may not adapt to the dynamic environment of Twitter when new concepts appear. Twevent filters out unnecessary events and thus has much fewer false alarms than our algorithm does. However, Twevent fails to detect many important events. One possible reason is that it introduces thresholds to identify bursty segments, which may not adapt well to various burst patterns of different types of events.



(a) Compare U_L (b) Compare *clu thres* (c) Compare V

Fig. 2. Compare event detection results with different parameter settings.

Table 2. Number of events detected by different methods on different days

	Nov 1	Nov 2	Nov 3	Nov 4	Nov 5	Nov 6
TwiCal	2	0	2	3	3	0
Twevent ($n = 1$)	0	0	0	0	1	0
Twevent ($n \leq 2$)	0	0	0	0	1	0
Twevent ($n \leq 3$)	1	0	0	0	1	0
Our Method	6	4	5	5	10	5

[2] http://statuscalendar.com.

Impact of Parameter Settings. We first compare the impact of U_L, clu_thres and V in Fig. 2. We set $U_L = 10$ min, $clu\_thres = 0.8$ and $V = 3$ if not specifying their varying values.

In Fig. 2a, we present the number of events detected with U_L =5 min and 10 min. We can see that more events are detected with a shorter time unit. However, the drawback of using a shorter time unit is that we report events more frequently and thus may return more false alarm events.

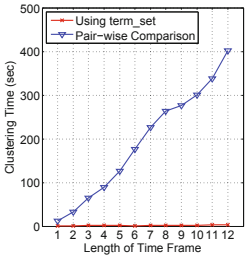
We then compare how the clustering threshold affects the results. In Fig. 2b, we present the number of events detected with clu_thres =0.6, 0.8 and 1, respectively. The results show that a lower clu_thres yields better performance. However, we observe that the clusters generated by a lower clu_thres are more messed up with words referring to different events.

We finally evaluate how the number of past time windows we average on affects the results. The results in Fig. 2c indicate that our algorithm performs better with a smaller V. This may be that we could better adapt to the dynamic environment of Twitter when V is smaller.

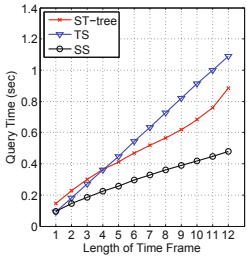
5.3 Efficiency Evaluation

We finally evaluate whether our algorithms are suitable for real-time application. Particularly, the preprocessing component takes on average 0.8ms to process a tweet, which is far beyond the average arrival rate of tweets from Twitter's sampled streaming API (52 ms/tweet). We present the results as follows.

Identification of Abnormal Terms, Clustering and Ranking. The time consumed by identification of abnormal terms depends on the total number of terms in a queried time window. Our results indicate that our algorithm identifies abnormal terms in 2.7 s when we average over 5 past time windows and the time window is as large as 120 min, demonstrating that we could efficiently complete the task in this step of processing. The time consumed by ranking also depends on the length of the queried time window. Results show that we can return top-20 clusters of a 120-min time window in 0.25 s.



(a) Clustering time with U_L = 10 min, $clu\ thres = 0.8$, $V = 3$



(b) Query time of ST-tree, TS and SS with $U_L = 10$ min, $V = 5$

Fig. 3. Compare clustering time consumption and Query time of ST-tree.

We finally come to the time consumption of the clustering component. In Fig. 3a, we present how our strategy of using *term_set* improves the clustering efficiency. Notice that the naive pair-wise comparison method is extremely inefficient when the length of queried time window is large. However, our algorithm could finish clustering for a 120-min time window in 4 s.

ST-tree Specific. In this part, we evaluate the efficiency of ST-tree. Adjustable time window event detection differs from the original problem in three aspects: built-up of ST-tree, update and query of statistics. For the built-up of ST-tree, we can construct a complete ST-tree with 360 leaves ($L_M = 60$ and $V = 5$) in 0.032 s, showing that initialization of ST-tree is efficient.

Table 3. Update Time of ST-tree and SS (U_L=10 min)

Algorithm	L_M	V	# of Leaves	Update Time (sec)
ST-tree	6	5	36	0.0325
SS	6	5	-	0.4932
ST-tree	30	5	180	0.1764
SS	30	5	-	2.0573
ST-tree	30	10	330	0.3584
SS	30	10	-	2.2059

In Table 3, we present the update time of ST-tree and SS with different values of L_M and V. We do not compare with TS since TS only needs to update statistics of the current time unit, which is performed continuously in the preprocessing component. We can see that ST-tree takes much less time than SS to update statistics. Even when updating a ST-tree with 330 leaves, we finish the procedure in less than 0.4s, which is highly efficient.

In Fig. 3b, we present the query time of three algorithms w.r.t. various queried time window lengths L. We set the time unit as 10 min, and L_M as 12 time units, i.e. 2 h. The first observation is that SS is the most efficient in all lengths of queried time window due to its $O(V)$ query time complexity. Second, when the length of queried time window is small, i.e. less than 4, TS beats ST-tree. The reason is that TS simply aggregates statistics of L time units, while ST-tree may need to visit some deep nodes of ST-tree to obtain statistics of a short time window and thus result in a longer query time than TS.

6 Related Work

A comprehensive survey [2] on event detection on Twitter is proposed. We summarize major techniques of event detection on Twitter. Many previous works focused on detecting specific category of events [6,8,10,12], or required queries from users and assumed a topic-based stream [12]. Recently, some works

focused on detecting topic-based events in crowdsourcing markets and social networks [16,17] and assisted task assignment in the markets and social networks [13,14,18,19]. In addition, [11] claimed to be the first to detect open-domain events on Twitter. Some others [5,20] could also detect unspecific events. However, [5,11,20] used time-demanding techniques and were not suitable for real-time applications on Twitter. Some works detected only geo-spatial events [1]. Furthermore, the issue of detecting experts on Twitter is also studied [3,4].

To detect emerging events, some works first identified bursty words [1,6,7,9]. However, they often introduced burst thresholds. For grouping terms/tweets into events, two types of techniques were developed: clustering [1,6,9] and graph partitioning [20]. Clustering of short-text tweets could be difficult, while graph partitioning based methods may be time-consuming.

Finally, adjustable time window event detection is not yet studied. [1,7] both used fixed time frames to detect bursty terms. [15] enabled timeline zooming.

7 Conclusion

In this paper, we study the problem of adjustable time window based event detection in Twitter. We design an abnormality score calculated by term frequency, user frequency and past averaged records to identify abnormal terms in a simple and free-of-threshold way. We then propose a highly efficient clustering algorithm to group co-occurring terms to form clusters, i.e. events. We further design ST-tree to adjustable time window based enable event detection. Extensive experiments on real dataset show that our algorithm can effectively and timely identify real-world events.

Acknowledgment. This work is supported in part by the National Science Foundation of China (NSFC) under Grant No. 61502021, 61328202, and 61532004, National Grand Fundamental Research 973 Program of China under Grant 2012CB316200, the Hong Kong RGC Project N_HKUST637/13, NSFC Guang Dong Grant No. U1301253, Microsoft Research Asia Gift Grant, Google Faculty Award 2013.

References

1. Abdelhaq, H., Sengstock, C., Gertz, M.: Eventweet: online localized event detection from twitter. PVLDB **6**(12), 1326–1329 (2013)
2. Atefeh, F., Khreich, W.: A survey of techniques for event detection in twitter. Comput. Intell. **31**(1), 132–164 (2013)
3. Cao, C.C., She, J., Tong, Y., Chen, L.: Whom to ask?: jury selection for decision making tasks on micro-blog services. PVLDB **5**(11), 1495–1506 (2012)
4. Cao, C.C., Tong, Y., Chen, L., Jagadish, H.V.: Wisemarket: a new paradigm for managing wisdom of online social users. In: SIGKDD 2013, pp. 455–463 (2013)
5. Cataldi, M., Di Caro, L., Schifanella, C.: Emerging topic detection on twitter based on temporal and social terms evaluation. In: MDMKDD 2010, p. 4 (2010)
6. Goorha, S., Ungar, L.: Discovery of significant emerging trends. In: SIGKDD 2010, pp. 57–64 (2010)

7. Li, C., Sun, A., Datta, A.: Twevent: Segment-based event detection from tweets. In: CIKM 2012, pp. 155–164 (2012)
8. Li, R., Lei, K.H., Khadiwala, R., Chang, K.C.: Tedas: a twitter-based event detection and analysis system. In: ICDE 2012, pp. 1273–1276 (2012)
9. Mathioudakis, M., Koudas, N.: Twittermonitor: trend detection over the twitter stream. In: SIGMOD 2010, pp. 1155–1158 (2010)
10. Popescu, A.M., Pennacchiotti, M.: Detecting controversial events from twitter. In: CIKM 2010, pp. 1873–1876 (2010)
11. Ritter, A., Etzioni, O., Clark, S., et al.: Open domain event extraction from twitter. In: KDD 2012, pp. 1104–1112 (2012)
12. Sakaki, T., Okazaki, M., Matsuo, Y.: Earthquake shakes twitter users: Real-time event detection by social sensors. In: WWW 2010, pp. 851–860 (2010)
13. She, J., Tong, Y., Chen, L.: Utility-aware social event-participant planning. In: SIGMOD 2015, pp. 1629–1643 (2015)
14. She, J., Tong, Y., Chen, L., Cao, C.C.: Conflict-aware event-participant arrangement. In: ICDE 2015, pp. 735–746 (2015)
15. Shou, L., Wang, Z., Chen, K., Chen, G.: Sumblr: continuous summarization of evolving tweet streams. In: SIGIR 2013, pp. 533–542 (2013)
16. Tong, Y., Cao, C.C., Chen, L.: Tcs: Efficient topic discovery over crowd-oriented service data. In: SIGKDD 2014, pp. 861–870 (2014)
17. Tong, Y., Cao, C.C., Zhang, C.J., Li, Y., Chen, L.: Crowdcleaner: Data cleaning for multi-version data on the web via crowdsourcing. In: ICDE 2014, pp. 1182–1185 (2014)
18. Tong, Y., She, J., Ding, B., Wang, L., Chen, L.: Online mobile micro-task allocation in spatial crowdsourcing. In: ICDE 2016 (2016)
19. Tong, Y., She, J., Meng, R.: Bottleneck-aware arrangement over event-based social networks: the max-min approach. World Wide Web Journal (to appear)
20. Weng, J., Lee, B.S.: Event detection in twitter. In: ICWSM 2011, pp. 401–408 (2011)