Erschienen in: Advanced Information Systems Engineering : 27th International Conference, CAiSE 2015, Stockholm, Sweden, June 8-12, 2015, Proceedings / Zdravkovic, Jelena et al. (Hrsg.). - Cham : Springer, 2015. - (Lecture Notes in Computer Science ; 9097). - S. 35-49. - ISSN 0302-9743. - eISSN 1611-3349. - ISBN 978-3-319-19068-6
https://dx.doi.org/10.1007/978-3-319-19069-3_3

Run-Time and Task-Based Performance of Event Detection Techniques for Twitter

Andreas Weiler(✉), Michael Grossniklaus, and Marc H. Scholl

Department of Computer and Information Science,
University of Konstanz, P.O. Box 188, 78457 Konstanz, Germany
{andreas.weiler,michael.grossniklaus,marc.scholl}@uni-konstanz.de

Abstract. Twitter's increasing popularity as a source of up to date news and information about current events has spawned a body of research on event detection techniques for social media data streams. Although all proposed approaches provide some evidence as to the quality of the detected events, none relate this task-based performance to their run-time performance in terms of processing speed or data throughput. In particular, neither a quantitative nor a comparative evaluation of these aspects has been performed to date. In this paper, we study the run-time and task-based performance of several state-of-the-art event detection techniques for Twitter. In order to reproducibly compare run-time performance, our approach is based on a general-purpose data stream management system, whereas task-based performance is automatically assessed based on a series of novel measures.

Keywords: Event detection · Performance evaluation · Twitter streams

1 Introduction

With 271 million monthly active users[1] that produce over 500 million tweets per day[2], Twitter is the most popular and fastest-growing microblogging service. Microblogging is a form of social media that enables users to broadcast short messages, links, and audiovisual content. In the case of Twitter, these so-called *tweets* can contain 140 characters and are posted to a network of *followers* as well as to a user's public timeline. The brevity of tweets make them an ideal mobile communication medium and Twitter is therefore increasingly used as an information source for current events as they unfold. For example, Twitter data has been used to detect earthquakes [17], to track epidemics [10], or to monitor elections [21].

In this context, an *event* is defined as a real-world occurrence that takes place in a certain geographical location and over a certain time period [3]. For traditional media such as newspaper archives and news websites, the problem of event

[1] http://www.statista.com/study/9920/twityter-statista-dossier/
[2] http://www.sec.gov/Archives/edgar/data/1418091/000119312513390321/d564001ds1.htm

detection has been addressed by research from the area of Topic Detection and Tracking (TDT). However, topic detection in Twitter data streams introduces new challenges. First, Twitter "documents" are much shorter than traditional news articles and therefore harder to classify. Second, tweets are not redacted and thus contain a substantial amount of spam, typos, slang, etc. Finally, the rate at which tweets are produced is very bursty and continually increases as more people adopt Twitter every day.

Several techniques for event detection in Twitter have been proposed. However, most of these approaches suffer from two major shortcomings. First, they tend to focus exclusively on the information extraction aspect and often ignore the streaming nature of the input. As a consequence, they make unrealistic assumptions, which limit their practical value. Examples of such assumptions include buffering entire months of Twitter data before processing it or fixing a complex set of parameters at design-time using sample data. Second, very few authors have evaluated their technique quantitatively or comparatively. While most provide some qualitative evidence demonstrating their task-based performance, very few consider run-time performance. Therefore, little or no research to date has measured the computing cost of the same result quality for different approaches. We argue that understanding this trade-off is particularly important in a streaming setting, where processing needs to happen in real-time.

In this paper, we present a method to study the task-based and the run-time performance of current and future event detection techniques. In order to measure comparable run-time performance numbers, we propose to "standardize" event detection techniques by implementing them based on a single data stream management system. Additionally, we developed several scalable measures to assess the task-based performance of event detection techniques automatically, i.e., without painstakingly crafting a gold standard manually. The specific contributions of this paper are as follows.

1. Streaming implementations of state-of-the-art event detection techniques for Twitter that are consistent with respect to each other.
2. Detailed study of the task-based and run-time performance of well-known event detection techniques.
3. Platform-based approach that will enable further systematic performance studies for novel event detection techniques in the future.

The remainder of this paper is structured as follows. Section 2 provides the background of this work by summarizing the state of the art in event detection for Twitter data streams. In Sect. 3, we give a brief overview of Niagarino, the data stream management system that we used as an implementation platform. Section 4 describes the selected event detection techniques and their streaming implementations using Niagarino. Section 5 presents the results of the evaluation that we performed in order to study the selected task-based and run-time performance of these event detection techniques. Finally, concluding remarks are given in Sect. 6.

2 Background

Our work is situated in the research field of analysis and knowledge discovery for social media data. Bontcheva *et al.* [8] provides a good general overview of sense making of social media data by surveying state-of-the-art approaches for mining semantics from social media streams. Due to the fast propagation speed of information in social media networks, a large number of works focus on event or topic detection and tracking for various domains. In this setting, Farzindar and Khreich [11] surveyed techniques for event detection in Twitter. The work presented in this paper targets approaches that support the detection of general (unknown) events [3] and we will therefore focus the following discussion on approaches that share this goal.

Petrović *et al.* [16] propose to use an online clustering approach that is based on locality sensitive hashing. The approach uses the number of tweets and hashtags, but also introduces a novel measure of entropy for the analysis. The method was evaluated using six months of data containing 163.5 million tweets and an average precision score was calculated against a manually labeled result set. Becker *et al.* [6] present an approach for "real-world event detection on Twitter" that uses an online clustering method in combination with a support vector machine classifier. They focus on hashtags with special capitalization and check for retweets, replies, and mentions. The method is evaluated against a manually labeled result set for a one-month data set with 2.6 million tweets. Long *et al.* [14] use divisive clustering, whereas Weng and Lee [21] use discrete wavelet analysis and graph partitioning. Both of these approaches use word frequencies of individual words for event detection. The latter approach was evaluated by using a self-built ground truth, which is prepared by using a latent dirichlet allocation (LDA) method [7]. Cordeiro [9] proposes the use of continuous wavelet analysis to detect event peaks in the signal of hashtags and summarizes the detected events by using LDA. For evaluation purposes, they used a visual illustration of their results obtained from an eight-day data set with 13.6 million tweets. Zimmermann *et al.* [22] present a text stream clustering method that detects, tracks, and updates large and small bursts in a two-level (global and local) topic hierarchy by using collected news articles. The technique proposed by enBloque [4] to detect emergent events relies on statistics about tags and pairs of tags. These statistics are computed using a time-sliding window and monitored for shifts in order to capture unpredictable and thus interesting developments. It has been evaluated on a two-week Twitter data set by conducting experiments to measure run-time performance and a user study to assess task-based performance.

In summarizing the state of the art in event detection techniques for Twitter, it is important to note that all existing approaches are realized as custom ad-hoc implementations, which limits the reproducibility and comparative evaluation of their results. As a consequence, little to no comparative evaluations of different event detection methods exist. In particular, none of these approaches have been evaluated to relate their task-based (result quality) and run-time performance (tweets per second). Therefore, there is a comprehensive lack of evaluation methods for event detection techniques for social media data.

3 Niagarino Overview

In order to realize streaming implementations of state-of-the-art event detection techniques for Twitter, we use Niagarino[3], a data stream management system that is developed and maintained by our research group. The main purpose of Niagarino is to serve as an easy-to-use and extensible research platform for streaming applications such as the one presented in the paper. The concepts embodied by Niagarino can be traced back to a series of pioneering data stream management systems, such as Aurora [2], Borealis [1], and STREAM/CQL [5]. In particular, Niagarino is an offshoot of NiagaraST [13], with which it shares the most common ground. In this section, we briefly summarize the parts of Niagarino that are relevant for this paper.

In Niagarino, a query is represented as a directed acyclic graph $Q = (O, S)$, where O is the set of operators used in the query and S is the set of streams used to connect the operators. The Niagarino data model is based on relational tuples that follow the first normal form, i.e., have no nesting. Two types of tuples can be distinguished, data and metadata tuples. Data tuples are strongly typed and have a schema that defines the domains of all attributes. All data tuples in a stream share the same schema, which corresponds to the output schema of the operator that generates the tuples and must comply with the input schema of the operator that consumes the tuples. In contrast, metadata tuples, so-called messages, are untyped and typically self-describing. Therefore, different messages can travel in the same stream. Messages are primarily used to transmit data and operator statistics in order to coordinate the operators in a query. Each stream is bidirectional consisting of a forward and a backward direction. While data tuples can only travel forward, messages can travel in both directions.

Based on its relational data model, Niagarino implements a series of operators. The selection (σ) and projection (π) operator work exactly the same as their counterparts in relational databases. Other tuple-based operators include the derive (f) and the unnest (μ) operator. The derive operator applies a function to a single tuple and appends the result value to the tuple. The unnest operator splits a "nested" attribute value and emits a tuple for each new value. A typical use case for the unnest operator is to split a string and to produce a tuple for each term it contains. Apart from these general operators, Niagarino provides a number of stream-specific operators that can be used to segment the unbounded stream for processing. Apart from the well-known time and tuple-based window operators (ω) that can be tumbling or sliding [12], Niagarino also implements data-driven windows, so-called frames [15]. Stream segments form the input for join ($\bowtie$) and aggregation (Σ) operators. As with derive operators, Niagarino also supports user-defined aggregation functions. Niagarino operators can be partitioned into three groups. The operators described above are general operators, whereas source operators read input streams and sink operators output results. Each query can have multiple source and sink operators.

[3] http://www.informatik.uni-konstanz.de/grossniklaus/software/niagarino/

This classification is similar to the notion of spouts and bolts used in Twitter's data stream management system Storm [19].

Niagarino is implemented in Java 8 and relies heavily on its new language features. In particular, anonymous functions (λ-expressions) are used in several operators in order to support lightweight extensibility with user-defined functionality. The current implementation runs every operator in its own thread. Operator threads are scheduled implicitly using fixed-size input/output buffers and explicitly through backwards messages.

4 Event Detection Techniques

We focus on techniques with the specific task of first story detection, i.e., the detection of general (unknown) events, which is defined as a subtask of TDT [3]. In this section, we briefly describe the five state-of-the-art techniques that we selected for our study in terms of their functionality and the parameters used. Figure 1 illustrates these techniques by means of Niagarino query plans that use the operators described in the previous section. As can be seen in the figure, all of these techniques use the same pre-processing steps before the streaming tuples enter the actual event detection phase. The pre-processing selects all tweets that are non-retweets and in English. Additionally, each tuple is enriched with the derived distinct terms of the tweet that are not contained in a standard English stop-word list or can be considered noise (e.g., less than three characters, unknown characters, repetition of the same pattern, or terms without vowels).

The *TopN* algorithm assigns each individual term a single value based on the inverse document frequency (IDF) [18] over an entire time window. All values are then sorted and the top n terms are reported as events together with their top m most frequently co-occurring terms, which are also obtained by using the IDF measure.

The *Latent Dirichlet Allocation (LDA)* [7] is a hierarchical Bayesian model that explains the variation in a set of documents in terms of a set of n latent "topics", i.e., distributions over the vocabulary. Since LDA is normally used for topic modeling, we equate a topic to an event. For each time window, LDA extracts n events that are described by m terms. The parameter i defines the number of iterations performed in the modeling phase, where a higher value typically increases the quality of the detected events. To perform the LDA, we use *Mallet*[4], an existing Java library.

Our own *Shifty* [20] technique calculates a measure that is based on the shift of IDF values of single terms in pairs of successive sliding windows of a pre-defined size. First, the IDF value of each term in a single window (with size s_{input}) is continuously computed and compared to the average IDF value of all terms within that window. Terms with an IDF value above the average are filtered out. The next step builds a window with size s_1 that slides with range r_1 in order to calculate the shift from one window to the next. In this step,

[4] http://mallet.cs.umass.edu

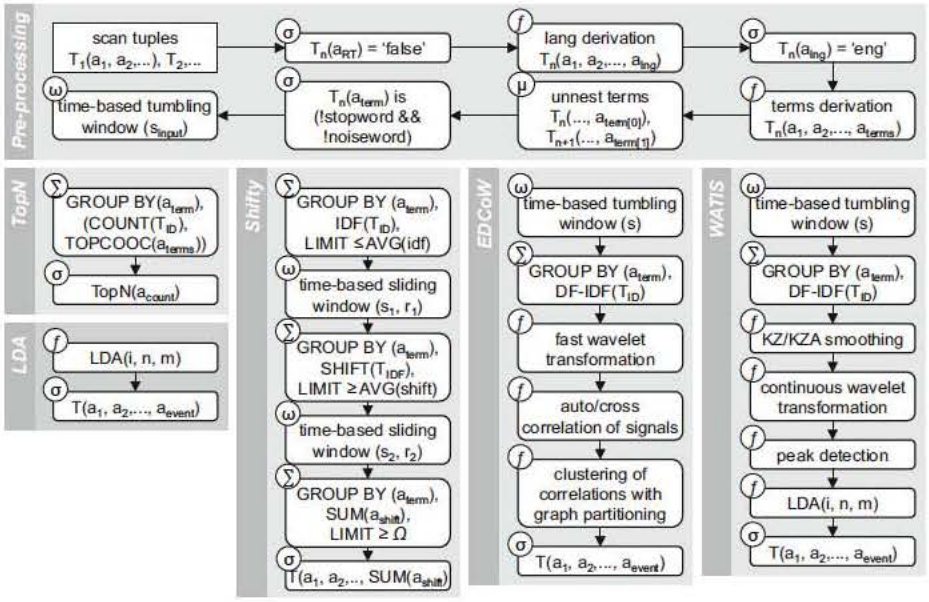


Fig. 1. Niagarino query plans of the five selected event detection techniques

the shift value is again checked against the average shift of all terms and only terms with a shift above the average are retained. In the last step, a new sliding window with size s_2 that slides with range r_2 is created. The total shift value is computed as the sum of all shift values of the sub-windows of this window. If this total shift value is greater than the pre-defined threshold Ω, the term is detected as event and reported together with its top 4 co-occurring terms.

The first step of the *Event Detection with Clustering of Wavelet-based Signals (EDCoW)* [21] algorithm is to partition the stream into intervals of s seconds and to build DF-IDF signals for each distinct term in the interval. These signals are further analyzed using discrete wavelet analysis that builds a second signal for the individual terms. Each data point of this second signal summarizes a sequence of values from the first signal with length Δ. The next step then filters out trivial terms by checking the corresponding signal auto-correlations against a threshold γ. The remaining terms are then clustered to form events with a modularity-based graph partitioning technique. Insignificant events are filtered out using a threshold parameter ϵ. Since this approach detects events with a minimum of two terms, we introduced an additional enrichment step that adds the top co-occurring terms to obtain events with at least five terms.

The *Wavelet Analysis Topic Inference Summarization (WATIS)* [9] algorithm also partitions the stream into intervals of s seconds and builds DF-IDF signals for each distinct term. Due to the noisy nature of the Twitter data stream, signals are then processed by applying the adaptive Kolmogorov-Zurbenko filter (KZA), a low-pass filter that smoothens the signal by calculating a moving average with

i_{kz} iterations over n intervals. It then uses continuous wavelet transformation to construct a time/frequency representation of the signal and two wavelet analyses, the tree map of the continuous wavelet extrema and the local maxima detection, to detect abrupt increases in the frequency of a term. To enrich events with more information, the previously mentioned LDA algorithm (with i_{lda} iterations) is used to finally report events that consist of five terms each.

5 Evaluation

The evaluation of event detection techniques is itself a challenging task. Determining an F_1 score in terms of precision and recall would require a ground truth (gold standard) to which the detected events can be compared. Due to the lack of such a ground truth for the Twitter data stream, some existing approaches have been evaluated using a manually created ground truth or based on user studies, if at all. Since both of these methods are very time-consuming and do not scale, we have experimented with a number of measures that can be applied automatically. In this section, we discuss the motivation behind these measures and present detailed results that were obtained by using them.

5.1 Measures

In order to evaluate different techniques automatically, we defined five main measures (some with sub-measures), which are used for the individual ratings. The measures are described in the following.

Precision (Search Engine). This measure describes the percentage of events that can be verified with the use of a search engine (www.google.com). For each detected event, the search engine is queried using the five event terms and a specific date range. A rating between 1 and 10 (*GoogleN*) is computed by checking how many of the first ten result hits point to a news website. News websites are identified based on a whitelist of domain names containing sites such as CNN, CBS, Reuters, NYTimes, and the Guardian. Based on this measure, detected events can be rated with respect to their newsworthiness on or at least one day after the detection date.

Precision (DBPedia). This measure is calculated using the DBPedia[5] data set, which contains the abstracts (long versions) from all Wikipedia articles. In order to query the roughly four million English abstract, the native XML database BaseX[6] is used. For each detected event, the number of matching abstracts in DBPedia is computed using XQuery Full Text. We have defined three sub-measures. *DBPedia5* is the precision using all five event terms, *DBPedia4S* only uses the top four event terms, and *DBPedia4A* queries DBPedia with all subsets of cardinality four. For the first two measures, an abstract is considered a match to an event if it contains *all* terms that were used in the query. For the third measure, an abstract matches if it contains all terms of *one* of the combinations.

[5] http://dbpedia.org/
[6] http://basex.org

Recall. In order to compute the recall, *Bloomberg*[7] was crawled as their archive maintains a list of the most important news articles for each day. Crawling individual days leads to an average of about 200 events per day. Each crawled news item is then tokenized and cleaned by the same processes as the tweets. As a consequence, the short description of each news item by a series of terms can be very similar to the one obtained from the tweets. In order to calculate the similarity between detected events and a news item, $\mathbf{eventSim}(e_1, e_2)$ is used, which is based on the Levenshtein distance.

$$\mathbf{levSim}(t_1, t_2) = 1.0 - \text{lev}(t_1, t_2) \;/\; \max(\{|t_1|, |t_2|\}) \tag{1}$$

$$\mathbf{termSim}(t_1, t_2) = \begin{cases} 0 & \text{levSim}(t_1, t_2) < \mathit{minTermSim} \\ 1 & \text{otherwise} \end{cases} \tag{2}$$

$$\mathbf{eventSim}(e_1, e_2) = \frac{1}{N} \sum_{i=0,j=0}^{N} \text{termSim}(e_1[t_i], e_2[t_j]) \tag{3}$$

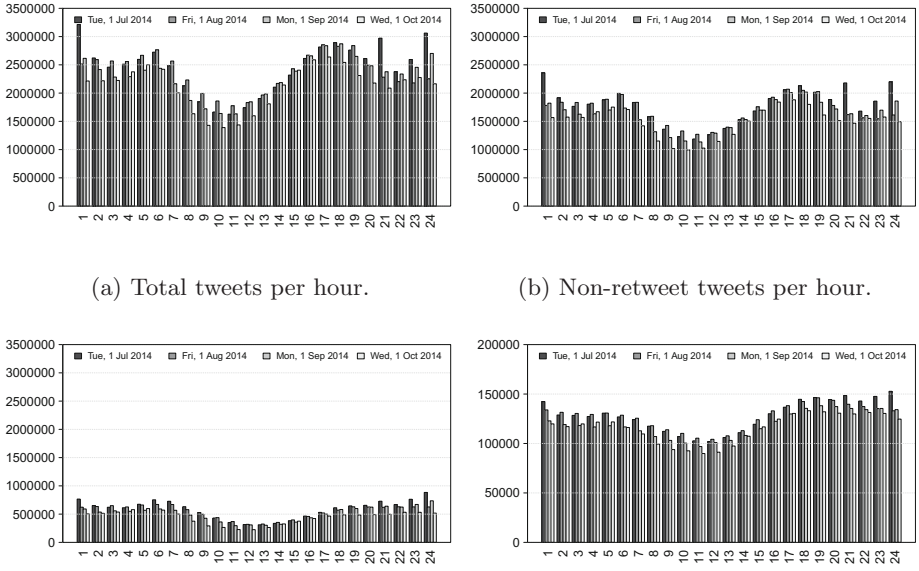
The motivation behind $\mathbf{eventSim}(e_1, e_2)$ is to compensate for misspellings or alternate spellings of terms as well as for different term sets describing similar events. An event is represented as an alphabetically sorted list of terms $e = [t_0, \ldots, t_n]$. Each term $t_1 \in e_1$ is compared to each term $t_2 \in e_2$ using the $\mathbf{levSim}(t_1, t_2)$, which is the Levenshtein distance normalized to the range $[0 \ldots 1]$. If the similarity of a term of e_1 to a term of e_2 is above the threshold *minTermSim*, this combination is marked as hit and the algorithm continues with the next term of e_1. Finally, $\mathbf{eventSim}(e_1, e_2)$ aggregates the number of hits and normalizes it with the number of terms.

In an effort to obtain a reasonable amount of hits, the parameters of this formula are set rather low. The parameter *minTermSim* is set to 0.7 and the overall limit for **eventSim** is set to 0.2. Two sub-measures are defined for the recall. *Bloom1D* calculates the recall just for the given date, whereas *Bloom2D* also includes the following day.

Duplicate Event Detection Rate (DEDR). This measure is also based on the event similarity defined above in order to calculate the similarity of the events for one single technique and data set. Two sub-measures have been defined. For *ADEDR* (almost duplicate event detection rate) the parameter *minTermSim* is set to 0.8 and the limit for *eventSim* is set to 0.5, whereas for *FDEDR* (full duplicate event detection rate) the *minTermSim* is the same but the limit for *eventSim* is set to 0.9.

Run-time Performance. Run-time performance is measured as the number of tweets per second that a technique is able to process.

[7] http://www.bloomberg.com/archive/news/



(a) Total tweets per hour.

(b) Non-retweet tweets per hour.

(c) English non-retweet tweets per hour.

(d) Distinct terms per hour.

Fig. 2. Statistics of the Twitter data set

5.2 Data Sets

The data sets used in the study presented in this paper consist of 10% of the public live stream of Twitter for four days. Using the Twitter Streaming API[8] with the so-called "Gardenhose" access level, which is a randomly sampled substream, we collected data for the first day of June, August, September, and October. Figure 2 provides statistics of the initial data set as well as for the processing steps that are common to all techniques (cf. Fig. 1). Figure 2a presents the total number of tweets for the chosen days grouped by the hour (given in GMT+1). As can be seen, the rate of tweets follows a regular daily pattern. On average, the incoming stream contains 2.3 million tweets/hour and 35,000 tweets/minute. Figure 2b shows the hourly tweet volumes after filtering out retweets at an average of 1.6 million tweets/hour. After the next step, shown in Fig. 2c, the data sets are further reduced to an average of 500,000 tweets/hour by filtering out tweets that are not in English. Finally, Fig. 2d shows an average of 120,000 distinct terms/hour that have been derived from all English tweets.

5.3 Experimental Setup

In order to be able to compare the results of the five chosen techniques in a fair way, they have to be aligned in terms of the rate and number of events detected.

[8] https://dev.twitter.com

Table 1. Average number of detected events per techniques and dataset

		dataset				
		Jul1	**Aug1**	**Sep1**	**Oct1**	**AVG**
technique	**Top15**	360	360	360	360	360
	LDA500	360	360	360	360	360
	Shifty	327	316	354	402	350
	EDCoW	353	375	396	409	383
	WATIS	270	261	287	276	273

The rate can be controlled by setting the time window on which a technique is performed. Since we are interested in (near) real-time event detection, a window of one hour was used. Note, that *Shifty* is the only true streaming algorithm that reports results continuously, whereas all other techniques only produce results after each hour. The number of events that are detected can be controlled by setting the specific parameters of each technique. Given that our recall measure assumes an average of 200 events per day and compensating for events that are detected multiple times, we aim for about 350 events per day. The parameter settings used are described below, whereas the actual number of detected events per day and technique are shown in Tab. 1.

TopN. Per hour, the top $n = 15$ events are reported together with $m = 5$ co-occurring terms to obtain a total of 360 events per day

LDA. LDA is set to perform $i = 500$ iterations and to report 15 events, described by $m = 5$ terms each, per hour, yielding again a total of 360 events per day.

Shifty. The IDF value is calculated over 1-minute intervals. The size of the window used to compute the IDF shift is $s_1 = 2$ minutes. The size of the window that aggregates and filters the IDF shift is $s_2 = 4$ minutes. Both windows slide by range $r_1 = r_2 = 1$ minute. By setting the threshold $\Omega = 0.35$, we obtain all terms with a minute by minute IDF value that increases more than 35% over four minutes.

EDCoW. The size of the initial intervals is set to $s = 10$ seconds and the number of intervals that are combined by the wavelet analysis to $\Delta = 32$, yielding a total window size per value of 320 seconds. The other parameters are set to the same values as in the original paper ($\gamma = 1$ and $\epsilon = 0.2$). As the original paper fails to mention the wavelet type that was used, we experimented with several types. The results reported in this paper are based on the *Discrete Meyer* wavelet, which showed the best performance.

WATIS. The length of initial intervals is set to $s = 85$ seconds. For the KZ/KZA analysis, $n = 5$ intervals and $i_{kz} = 5$ iterations are used, yielding a total window size of 425 seconds. LDA is set to perform $i_{lda} = 500$ iterations and report a description with five terms per detected event.

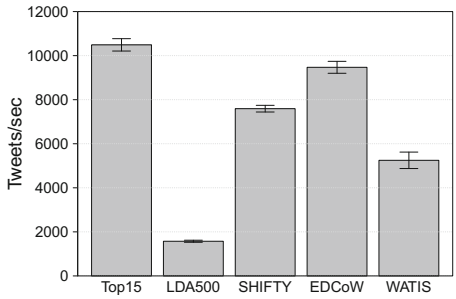


Fig. 3. Average run-time performance

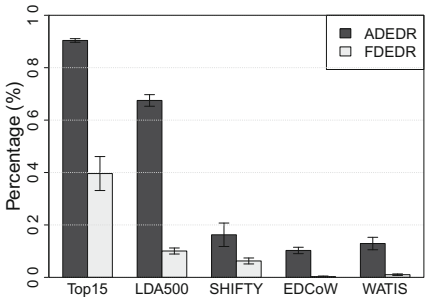


Fig. 4. Average duplicate rate

5.4 Results

In the following, we present the results of our evaluation of event detection techniques in terms of run-time and task-based performance. Rather than discussing all results that we have obtained, we focus on the most significant measures and outcomes. While we do not claim that our measures are absolute, it should be noted that these results support relative conclusions.

Run-Time Performance. Run-time performance was measured using Oracle Java 1.8.0_25 (64 bit) on server-grade hardware with 2 Intel Xeon E5345s processors at 2.33 GHz with 4 cores each and 24 GB of main memory. The corresponding results for all techniques in terms of throughput (tweets/second) are given in Fig. 3. We note that the performance of all techniques is very stable across the four days for which experiments were run. Taking into account the average rate of 35,000 tweets/minute (583 tweets/second), we can derive that all techniques are able to process the 10% stream in real-time on the tested hardware. However, taking a 100% stream ($\sim 5,830$ tweets/second) into account, both *LDA500* and *WATIS* would be too slow to process the stream in real-time on the tested hardware. In both techniques, the number of LDA iterations could be reduced, i.e., trading off result quality for performance. Finally, we point out that our experimental setup is stacked against our own technique, *Shifty*. In contrast to the other approaches that can only process tweets at the end of each one-hour window, *Shifty* processes tweets continuously and can therefore amortize its processing cost over the one-hour window.

Task-Based Performance. The first measure of task-based performance that we will examine is the duplicate event detection rate. Results obtained using both the *ADEDR* and *FDEDR* sub-measures are given in Fig. 4. In comparison to the other three techniques, both *Top15* and *LDA500* detect a large number of duplicates. This result is explained by the fact that these techniques identify events based on the absolute frequency of terms, i.e., without considering changes

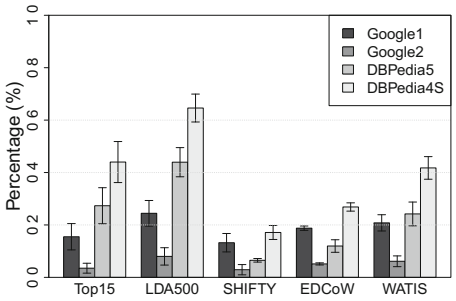


Fig. 5. Average precision

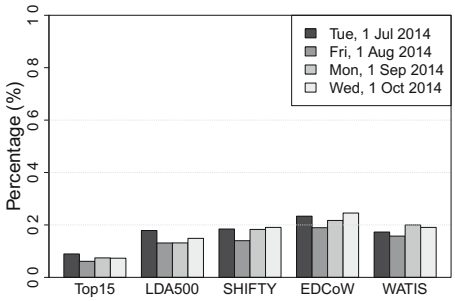


Fig. 6. Recall using *Bloom1D*

in the relative frequency. The *ADEDR* of the remaining three techniques is relatively low in the range of 15–18%. *Shifty*'s *FDEDR* stayed consistently below 10% in all our experiments, whereas *EDCoW* and *WATIS* do hardly detect any duplicates at all. Finally, the results also show that there is little deviation in the detected number of duplicates over the four days in our data set.

Apart from the duplicate event detection rate, we have also studied the task-based performance of the selected techniques in terms of precision and recall. Figure 5 summarizes the precision results of all techniques obtained with the *Google1*, *Google2*, *DBPedia5*, and *DBPedia4S* measures. We omit results from the *DBPedia4A* as our experiments showed that they are not discriminating. Even though the measures we defined yield a wide range of precision values, their relative ratio is always the same. Since our goal is to comparatively evaluate event detection techniques, we conclude that our measures are sound with respect to this criterion. Again, *Top15* and *LDA500* stand out with higher precision values than the other three techniques. The reason for this result is that our precision measures are slightly biased towards approaches that report duplicates.

Figure 6 shows the recall results for the *Bloom1D* measure. *Bloom2D* is omitted as the results are almost exactly the same. First of all, it can be seen from the figure that the recall of all techniques is relatively low at 10–20%. Note that our recall measure is based on the Bloomberg news website, which lists an average of 200 topics per day. Even though techniques were configured to report about 1.5× as many events, our recall measure is nevertheless ambitious. For example, it is difficult to imagine that enough people will tweet about a topic such as Heathrow's cargo statistics in order to detect it as an event. However, since we are only interested in relative measures, these low recall figures are not a problem. Rather, we can observe that *Top15* and *LDA500* generally have a lower recall than the other three techniques. As this outcome is to be expected due to the high duplicate event detection rate of these techniques, we can again conclude that our measure for recall is sound.

In order to summarize the most discriminating measures presented in this paper, we define three scoring functions that can be used to compare the run-time

and task-based performance of event detection techniques. The three scoring functions are defined as follows.

$$\textbf{FScore} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \tag{4}$$

$$\textbf{PFScore} = (\text{FScore} \times \text{performance}) \tag{5}$$

$$\textbf{DPFScore} = \text{PFScore} \times (1 - \mathit{DEDR}) \tag{6}$$

The first score, *FScore*, denotes the F_1 score that is calculated by using the value of the *Google1* and *Bloom1D* measures for precision and recall, respectively. Alternatively, using *DBPedia5* leads to very similar results. The second score, *PFScore*, also factors in the performance rate of the technique. Performance values are normalized to the range $[0 \ldots 1]$ by setting the maximum processing rate that we measured to 1. Finally, the last measure, *DPFScore*, also includes the duplicate event detection rate of the technique. In the following, we have used the value of the *FDEDR* measure to calculate *DPFScore*.

Based on these definitions, Fig. 7 shows the scores that were assigned to each of the five techniques as averages over the four days in the evaluation data set. Even though *Top15* scores relatively high in terms of precision, its *FScore* is low due to a poor recall because of duplicates. As *Top15* is consistently the fastest technique in our experiments, its *PFScore* is equal to its *FScore*. The high *DEDR* of *Top15* has a noticeable negative effect on its *DPFScore*. *LDA500*'s *FScore* is relatively high, but comes at a high performance penalty, which negatively affects both its *PFScore* and *DPFScore*. Based on these results, we can conclude that neither *Top15* nor *LDA500* are suitable event detection techniques. This result is not surprising as both of these techniques have originally not been developed for this task.

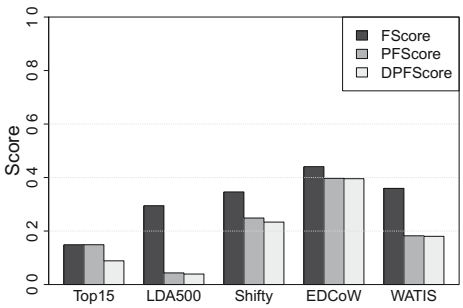


Fig. 7. Average rating scores

In contrast, the scores of *Shifty*, *EDCoW*, and *WATIS* are much better. In particular, none of these techniques suffer significantly from duplicate event detection. *Shifty* and *WATIS* have a similar *FScore*, but are both negatively affected by their performance score. However, since *Shifty*'s streaming algorithm was forced to an hourly reporting scheme for the sake of comparability, this score is still a good result for our technique. *EDCoW* scores impressive results for all scoring functions, which confirms that its status as the most cited event detection technique is well-deserved. This work however is the first to provide comparative and quantitative evidence for *EDCoW*'s quality.

Finally, we note that duplicate events are not always undesired, e.g., when tracking re-occurring events or changes in event descriptions. The need to study event detection techniques in both settings, motivates our separate definitions of *FScore*, *PFScore*, and *DPFScore*. Both *LDA500* and *Top15* could be extended

to explicitly avoid the detection of duplicate events. However, since the other techniques do allow for duplicates, we have chosen not to do so in this study.

6 Conclusion

In this paper, we addressed the problem of comparatively and quantitatively studying the task-based and run-time performance of state-of-the-art event detection techniques for Twitter. In order to do so, we have presented a two-pronged approach. First, we ensure comparable run-time performance results by providing streaming implementations of all techniques based on a data stream management system. Second, we propose several new measures that can assess the relative task-based performance of event detection techniques. The detailed study described in this paper has shown that these measures are sound and which of them are most discriminating. Finally, we defined scoring functions based on selected measures that revealed how the different techniques relate to each other as well as where their strengths and weaknesses lie.

As immediate future work, we plan to take advantage of our platform-based approach to study further techniques, e.g., enBloque [4] and the approach of Petrović *et al.* [16]. At the same time, the currently implemented techniques could be improved to process data continuously. Furthermore, the influence of the pre-processing on run-time and task-based performance should be studied. In our platform-based approach, we can easily remove existing operators (e.g., retweet filtering) and replace them with new operators (e.g., part-of-speech tagging or named-entity recognition). Finally, a deeper evaluation of how the different parameters of a technique influence the trade-off between run-time and task-based performance could give rise to adaptive event detection techniques.

Acknowledgments. We would like to thank our students Christina Papavasileiou and Harry Schilling for their contributions to the implementation of *WATIS* and *EDCoW*.

References

1. Abadi, D.J., Ahmad, Y., Balazinska, M., Çetintemel, U., Cherniack, M., Hwang, J., Lindner, W., Maskey, A., Rasin, A., Ryvkina, E., Tatbul, N., Xing, Y., Zdonik, S.B.: The design of the borealis stream processing engine. In: Proc. Intl. Conf. on Innovative Data Systems Research (CIDR), pp. 277–289 (2005)
2. Abadi, D.J., Carney, D., Çetintemel, U., Cherniack, M., Convey, C., Lee, S., Stonebraker, M., Tatbul, N., Zdonik, S.: Aurora: A New Model and Architecture for Data Stream Management. The VLDB Journal **12**(2), 120–139 (2003)
3. Allan, J.: Topic Detection and Tracking: Event-based Information Organization. Kluwer Academic Publishers (2002)
4. Alvanaki, F., Michel, S., Ramamritham, K., Weikum, G.: See what's enBlogue: real-time emergent topic identification in social media. In: Proc. Intl. Conf. on Extending Database Technology (EDBT), pp. 336–347 (2012)
5. Arasu, A., Babu, S., Widom, J.: The CQL Continuous Query Language: Semantic Foundations and Query Execution. The VLDB Journal **15**(2), 121–142 (2006)

6. Becker, H., Naaman, M., Gravano, L.: Beyond trending topics: real-world event identification on twitter. In: Proc. Intl. Conf on Weblogs and Social Media (ICWSM), pp. 438–441 (2011)
7. Blei, D.M., Ng, A.Y., Jordan, M.I.: Latent Dirichlet Allocation. J. Mach. Learn. Res. **3**, 993–1022 (2003)
8. Bontcheva, K., Rout, D.: Making Sense of Social Media Streams through Semantics: a Survey. Semantic Web **5**(5), 373–403 (2014)
9. Cordeiro, M.: Twitter event detection: combining wavelet analysis and topic inference summarization. In: Proc. Doctoral Symposium on Informatics Engineering (DSIE) (2012)
10. Culotta, A.: Towards detecting influenza epidemics by analyzing twitter messages. In: Proc. Workshop on Social Media Analytics (SOMA), pp. 115–122 (2010)
11. Farzindar, A., Khreich, W.: A Survey of Techniques for Event Detection in Twitter. Computational Intelligence (2013). http://dx.doi.org/10.1111/coin.12017
12. Li, J., Maier, D., Tufte, K., Papadimos, V., Tucker, P.A.: No Pane, No Gain: Efficient Evaluation of Sliding-Window Aggregates over Data Streams. SIGMOD Record **34**(1), 39–44 (2005)
13. Li, J., Tufte, K., Shkapenyuk, V., Papadimos, V., Johnson, T., Maier, D.: Out-of-Order Processing: A New Architecture for High-Performance Stream Systems. PVLDB **1**(1), 274–288 (2008)
14. Long, R., Wang, H., Chen, Y., Jin, O., Yu, Y.: Towards effective event detection, tracking and summarization on microblog data. In: Wang, H., Li, S., Oyama, S., Hu, X., Qian, T. (eds.) WAIM 2011. LNCS, vol. 6897, pp. 652–663. Springer, Heidelberg (2011)
15. Maier, D., Grossniklaus, M., Moorthy, S., Tufte, K.: Capturing episodes: may the frame be with you. In: Proc. Intl. Conf. on Distributed Event-Based Systems (DEBS), pp. 1–11 (2012)
16. Petrović, S., Osborne, M., Lavrenko, V.: Streaming first story detection with application to twitter. In: Proc. Conf. of the North American Chapter of the Association for Computational Linguistics (HLT), pp. 181–189 (2010)
17. Sakaki, T., Okazaki, M., Matsuo, Y.: Earthquake shakes twitter users: real-time event detection by social sensors. In: Proc. Intl. Conf. on World Wide Web (WWW), pp. 851–860 (2010)
18. Sparck Jones, K.: A Statistical Interpretation of Term Specificity and Its Application in Retrieval, pp. 132–142. Taylor Graham Publishing (1988)
19. Toshniwal, A., Taneja, S., Shukla, A., Ramasamy, K., Patel, J.M., Kulkarni, S., Jackson, J., Gade, K., Fu, M., Donham, J., Bhagat, N., Mittal, S., Ryaboy, D.V.: Storm @Twitter. In: Proc. Intl. Conf. on Management of Data (SIGMOD), pp. 147–156 (2014)
20. Weiler, A., Grossniklaus, M., Scholl, M.H.: Event identification and tracking in social media streaming data. In: Proc. EDBT Workshop on Multimodal Social Data Management (MSDM), pp. 282–287 (2014)
21. Weng, J., Lee, B.S.: Event detection in twitter. In: Proc. Intl. Conf on Weblogs and Social Media (ICWSM), pp. 401–408 (2011)
22. Zimmermann, M., Ntoutsi, I., Siddiqui, Z.F., Spiliopoulou, M., Kriegel, H.P.: Discovering global and local bursts in a stream of news. In: Proc. Symp. on Applied Computing (SAC), pp. 807–812 (2012)