Benchmarking general and specific event detection approaches

Sourabh Lal
École polytechnique fédérale de Lausanne
Lausanne, Switzerland
sourabh.lal@epfl.ch

June 9, 2017

Abstract

Social media such as Twitter is incredibly popular amongst its users to report real-life events. As millions around the world post about events taking place around them, Twitter has become a source for breaking news as users can often report on any incident before any mainstream media. This makes it immensely useful to have a 'sensory' tool that can detect events in real-time by parsing through a stream of tweets. This paper presents the results of my investigation of 4 techniques that can be employed to perform this task; 1) Analyzing feature trajectories - detecting the bursts of each feature, then clustering correlated features; 2) SigniTrend - keeping track of a significance measure for early detection, establishing a noise baseline, then clustering trending keyword pairs to singular events; 3) EDCoW - applying wavelet analysis on frequency based raw signals of words, following by filtering and clustering remaining words to form events; 4) Near Real-time Burst Topic Detection - optimizing a Vector space model with Local Sensitive Hashing, followed by bucketing and identifying bursts.

1 Introduction

Every second, on average, around 7,600 tweets are tweeted on Twitter, or which is equivalent to over 450,000 tweets every minute, or over 650 million every day. [2]. That is a huge amount of data published by users, and alot of information can be gained out of it. While it is true that there is a lot of tweets with unimportant information, there are alot of tweets posted by people about the events taking place around them. This information could prove immensely useful in case disaster - whether natural or man-made - strikes. Given the amount of updates posted in real time on social media like Twitter, there is the real potential to develop a system that can provide an alert in case of certain events. There are several challenges involved in event detection including a high volume of noise and very large datasets. Such challenges make event detection a non-trivial task, and therefore a robust and scalable solution is required. Several approaches have been proposed, each with varying merits, and in this paper we present how these techniques were tested and provide some metrics to illustrate my results.

This paper is organized as follows: The next section discusses each of the algorithms that are tested. Section 3, provides the details on the datasets used to test the algorithms. Section 4 discusses the implementation methods used for each algorithm and the tuning of parameters. And finally Section 5 provides a summary of the results.

2 Related Work

1. *Analyzing Feature Trajectories for Event Detection*[4] - This approach is developed on the premise that the majority of Topic Detection and tracking solutions are either to simplistic or too impractical due as there are too many parameters to tune. The paper describes an unsupervised event detection algorithm that borrows a well-known technique from signal processing: identifying distributional correlations among all its features by spectral analysis. The paper postulates that an event can be described by its representative features, and representative features of the same event share similar distributions. Furthermore, a feature can be included in several events with overlapping time frames and an important event has a largely reported set of representative features and vice versa for unimportant events.

2. *SigniTrend: Scalable Detection of Emerging Topics in Textual Streams by Hashed Significance Thresholds*[6] - This technique was developed as an attempt to solve 3 problems with event detection. Namely: differentiating between popular topics and trending topics, tracking interacting terms and scalability. The technique involves 3 steps: pre-processing, cyclical trend detection and end-of-epoch trend reporting. In order to determine whether a topic is trending, the algorithm checks how many standard deviations higher is the frequency of the word than its mean, and uses a weighted moving average and variance for continuous estimation as new tweets come in. The technique also uses a heavy-hitters hashing algorithm in order to increase scalability as there are too many terms and word pairs to keep track off, and not all of them are significant.

3. *EDCoW: Event Detection with Clustering of Wavelet-based Signals*[7] - EDCoW was designed specifically for event detection in Twitter, and it deals with the challenge of most twitter streams being noisy and therefore event detection algorithms are "overwhelmed by a flood of meaningless babbles". EDCoW works by using DF-IDF and applying wavelet transformation to build signals for the frequency of individual words. It filters away trivial words by computing the cross-correlation and filtering the signals with auto-correlation. It clusters the remaining words by using Newman's modularity based graph partitioning algorithm. Finally, it filters the clusters based on a significance threshold.

4. *Near Real-time Detection of Crisis Situations*[3] - This technique was developed to utilize Twitter data to promptly detect a situation that requires responsive action from a rescue crew. It is implemented by applying a Vector Space model as a First Story Detection method, and optimizes the results by using Locality Sensitive Hashing. For each tweet processed, this method returns a similarity score in [0.0 - 1.0] and the tweet that is most similar to it. The tweets are then clustered together based on the similarity score of each tweet's contents and the growth rate of the buckets in which similar tweets are collected.

3 Data Collection

In order to experiment with these event detection algorithms 4 different data sets were generated that each exhibit very different characteristics:

1. *ManchesterAttack* - Live data collected on the day of the Manchester attack using Twitter's Streaming API. This is by far the smallest dataset with

Table 1: Information about Datasets

Dataset	Tweets	Total words	Tokens	TimeSpan (buckets)
ManchesterAttack	1,537	18,102	3,985	1,581s (27 minutes)
3Attack	1,686	19,053	3,993	1810s (31 minutes)
MIBSample	152,540	1,454,372	80,183	42,019,774s(487 days)

only about 1500 tweets collected during the day. These tweets fall into 2 categories: Manchester attack specific tweets, padded by regular tweets. This was achieved using Twitter's required keyword filter when streaming: the Manchester attack tweets uses keywords such as 'manchester', 'uk', and 'terror', and the regular tweets use a random collection of words. The dataset is fairly evenly distributed. This can be considered a 'toy' dataset as it is designed such that there isn't much noise.

2. *3Attack* - Variation of ManchesterAttack with 3 distinct events. Each event appears at distinct points in the dataset, and with a different volume of tweets in each burst. (Nairobi Westgate attack = 100 tweets, Manchester attack = 50 tweets, Egypt bus attack = 25 tweets). In addition to these bursts there are regular tweets which contain some references to some of the key words.

3. *MIBSample* - This is a large dataset containing around 260,000 tweets obtained from MIB Projects as a sample of their larger dataset. It provides the tweets posted between 2007-2015 by 100 genuine twitter accounts as annotated by CrowdFlower contributors [1]. However the tweets are fairly sparse for most of the time span, and therefore to get a more realistic result only tweets from the last 2 years are considered.

4. *Combination of ManchesterAttack and MIBSample* - Various combinations of these two datasets was used as the tweets from MIBSample provided significant "noise" to the verified tweets from the ManchesterAttack dataset. To combine these data sets the timestamp of every tweet from MIBSample was translated forwards by 762 days to overlap the ManchesterAttack dataset.

5. *November15* - Collection of tweets from November 2015. This was one of the largest datasets available, however it was rather difficult to use as the tweets were very noisy, and a large percentage of tweets were in non-English characters making it hard to analyze the results. While it was used for early testing, it was discarded once the MIBSample dataset was available.

For testing variations of these data sets were used. These changes included removing certain words from certain time periods and injecting events to test if they are detected.

4 Methods

Many different techniques have been proposed for detecting bursts each with their advantages and disadvantages. In order to pick which ones to study for this project, one has to consider factors such as:

- Event detection on data collected in real time, vs on a static dataset
- Generic event detection vs domain specific event detection:
- Focus on scalability
- Industry standard approaches (for example TF-IDF and cosine similarity) vs novel approaches (for example techniques from signal processing)

- Single-word events vs clustering words to create an event.

Considering these different criteria, the 4 techniques highlighted in this paper provide a good balance of each of these factors in order to give a good general understanding of state-of-the-art approaches in the field.

For each technique listed, we read the time(in seconds since epoch) and text for every tweet. Pre-process the text as described below and enter it into the algorithm. These are the only 2 fields required.

4.1 Pre-Processing

Each of the algorithms except FeatureTrajectories requires that standard pre-processing is applied to the dataset before being passed into the algorithm. This includes removing urls, twitter account tags, hashtags, and emojis, tokenization, stemming and removal of stopwords, duplicates, and blank strings. Further pre-processing applied was the removal of symbols from the head and tail of tokens.

4.2 FeatureTrajectories

Feature trajectories doesn't require much configuration to run. It only requires a bucket size in seconds for implementing a term frequency. As all the classification is unsupervised there is nothing to configuration.

One of the key steps for FeatureTrajectories to work is being able to classify each feature (word) as important or unimportant. To do so, FeatureTrajectories compute the dominant power spectrum (S_f) of each feature and uses this as a score. In order to set the minimum threshold of an important event, the method requires that stop words are not removed during pre-processing, and uses the maximum dominant power spectrum of all the stopwords. This works under the assumption that all words with a higher S_f than all stopwords are important.

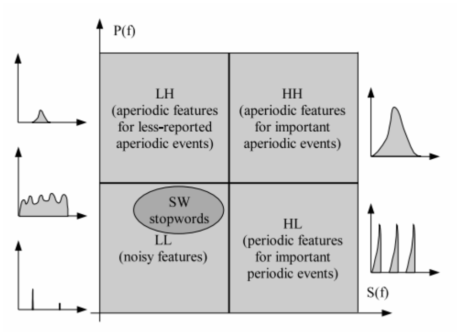
P(f)
LH (aperiodic features for less-reported aperiodic events)
HH (aperiodic features for important aperiodic events)
SW stopwords
LL (noisy features)
HL (periodic features for important periodic events)
S(f)

Figure 1: Classification of features [4]. Features with higher S_f than all stopwords go on the right, and features with dominant period (P_f) go on the top. After classification, features classified as SW or LL are discarded.

One can also vary the bucket size that is used for the term frequency. The effects of this vary by the distribution of tweets in the dataset, and the scale of the events that are being tracked.

4.2.1 Note about Pre-Processing

It should be noted that unlike the other algorithms, stop words are not removed as part of the pre-processing, but the algorithm removes them during its execution. However, apart from this, standard pre-processing is applied to the dataset before being passed into the algorithm.

4.3 SigniTrend

SigniTrend's implementation requires a bit of configuration for optimization. After testing SigniTrend with several datasets, and by manipulating these variables the following effects on the results were observed:

1. Window size: The amount of time steps the moving window shall consider. Varying the window size affects the scope when the significance score

is computed. As we want to know if an event is trending, its score reflects its relative popularity over the scope of the window. Therefore the window size should be selected based on the scope of the event we are looking for and how long it should be reported for.

2. Hash table bits: The amount of bits for our hash table. This did not seem to have a significant effect on the relatively small datasets in terms of significance score and even a 6-bit hash table is large enough to give an accurate score. However, it should be noted that increasing the hash table bits has an effect on memory and time. On a test on a subsection of the MIBSample injected with tweets about the Manchester attack, the following time and memory results were obtained:

Table 2: Running SigniTrend with varying hash table bits on 111,889 tweets across 6 months

Bits	Buckets	Memory	Time
6	64	422912B	1688s
8	256	1691648B	1865s
10	1024	6766592B	2439s

3. Hash function count: Refers to how many hash functions to use. If k hash functions are used, each word is mapped to at most k buckets. Tests conducted on suggested that there wasn't any impact on the results, though this may differ on much larger data sets with greater collisions. However it should be noted that the execution time increased linearly with the number of hash functions used.

4. Bias: Adjust to the expected level of background noise. This is a parameter that has a varying impact across all data sets depending on how noisy they were. This is shown in Figure 2. One of the difficulties with this approach is having to use trial and error to figure out the appropriate value of bias before executing the algorithm. Moreover, if the dataset is inconsistent this is a fixed value for the entire execution and cannot adapt based on the data being analyzed.

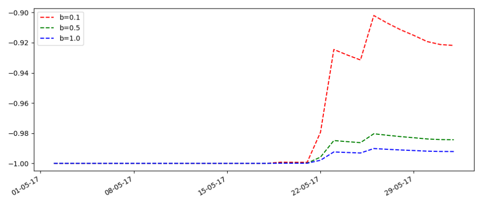

Figure 2: Effect on significance score of ‘Manchester’ across 33,856 tweets in a 30 day period by varying the bias b parameter

5. Alerting threshold: This refers to the threshold in units of standard deviations, above which frequencies are considered anomalously high. This is the most important parameter, as it determines which events are trending and which are not.

4.4 EDCoW

EDCoW doesn't require much configuration. The main parameters it uses are:

1. Delta: This refers to the time window size for the stage 2 signal within which we check if a word exhibits a burst.

2. Epsilon: This refers to the minimum significance score of an event. All events with a lower significance than this threshold are discarded. This threshold is computed by the algorithm

4.5 Near Real-time Burst Topic Detection

Near Real-time Burst Topic Detection requires a lot of calibration to get results, and is quite sensitive to the parameters. The following are the key parameters involved:

1. numBursts: This refers to the number of potential events we want to consider at any given time. Tweets which are similar are bucketed together, and this variable sets the maximum number of buckets we are storing.

2. maxbucketSize: Each bucket has a maximum size, this variable sets the maximum number of tweets that can together be considered an event.

3. similarityThreshold: This is used to filter out tweets whose similarity score is too low. If this is set too low, topics grouped in a bucket will be very specific. On the other hand, if it is set too high then the topics in the buckets will be diverse.

4. burstRate: This refers to the minimum threshold for classifying how fast a bucket is growing. If a bucket is growing faster than this variable, it is considered to be bursting. This can be set to any value between 0 and 1. It refers to the ratio between number of tweets in a bucket and how time since the oldest tweet in the bucket was tweeted.

5. burstCheckFrequency: This refers to how often we are checking for new bursting events. It can be set to any positive integer as it refers to the number of tweets that are processed between checks.

6. bucketSize: This refers to the bucket size that is used for the term frequency. The effects of this vary by the distribution of tweets in the dataset, and the scale of the events that are being tracked.

In the implementation of this technique, FSD using Locality Sensitive Hashing, Bucketing, and Identifying bursts were implemented. However the disaster detector, geo-localizing and alert component were not required for testing purposes. Moreover the technique was implemented locally, and not using any cloud based technologies.

4.6 Code

The code for the implementation of all the algorithms along with the datasets used for testing can be found here. Instructions on running the code and testing can be found in the repository README file.

5 Results and Evaluation

5.1 FeatureTrajectories

The algorithm was successfully able to detect 3 events in the ideal case example. Using a dataset that had sparse mentions of the words 'manchester' and 'attack' except for a small period of time when they occur densely. All 3 events detected make sense with regards to the dataset.

However it has to be noted that the results were not obtained with the standard implementation of FeatureTrajectories. As described in the method, FeatureTrajectories uses the maximum dominant power spectrum of all the stopwords as the minimum threshold when classifying important events. However, this does not work very well on Twitter data sets as many stopwords frequently occur in tweets, and their signals have a high dominant power spectrum. For example, in the illustration above, the three highest scoring key

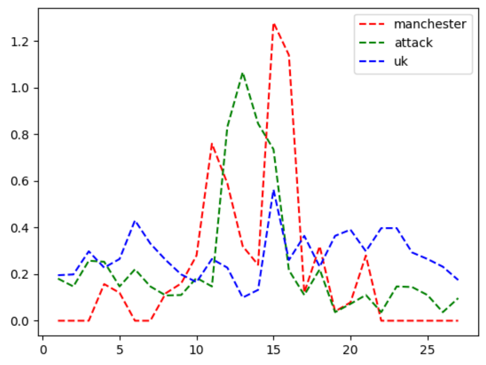

Figure 3: Feature Trajectory of keywords across time

Table 3: Testing on variation of ManchesterAttack seeded with 1 event

	Precision	Recall	Time
FeatureTrajectories[1]	1.00	1.00	3s
SigniTrend[2]	1.00	1.00	129s
EDCoW	0.00	0.00	104s[3]
RealTime[4]	0.13	0.80	41s

Table 4: Testing on variation of ManchesterAttack seeded with 3 events

	Precision	Recall	Time
FeatureTrajectories[1]	0.5	0.33	5s
SigniTrend[2]	1.00	0.33	142s
EDCoW	0.00	0.00	117s[3]
RealTime[4]	0.06	0.13	49s

Note: Duplicate events are considered as 1 event returned when calculating Precision and Recall

[1] = bucketSize = 60, FLAG=25

[2] = bucketSize = 60, bias = 0.1, alertingThreshold = 0.2

[3] = EDCoW using FastGreedy clustering (explained in further detail below)

[4] = Average across 5 runs as results were not always identical. bucketSize = 60, numBursts = 80, burstCheckFrequency = 10, maxbucketSize = 50, similarityThreshold = 0.5, burstRate = 0.5

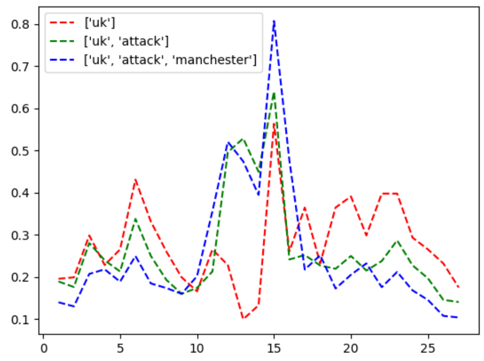

Figure 4: Visualization of events detected in a cleaned version of the ManchesterAttack dataset across time

words were ‘uk’, ‘attack’ and ‘manchester’ with scores of 56.707, 45.213, and 36.058 respectively, while stopwords like ‘the’ and ‘in’ had scores above 80. Therefore to obtain the above results, I manually set the threshold at a value such that the top 3-4 events filtered out. This is certainly impractical at a large scale as it requires alot of trial and error.

When it came to testing FeatureTrajectories with datasets with more than one event, results were promising by unclear. Three events were returned ([‘attack’, ‘uk’],[‘attack’, ‘uk’, ‘manchester’],[‘attack’, ‘uk’, ‘manchester’, ‘nairobi’]). The first 2 refer to the same event (the Manchester attack), however the 3rd event returned combines 2 of the attacks (”Manchester attack” and ”Nairobi attack”). Since these were 2 separate attacks it has to be marked as a false positive.

When testing FeatureTrajectories with data from MIB sample injected by ManchesterAttack, we are no longer able to detect these events. Given the larger volume of data, it was possible that there was another larger event occurring at around the same time period, however, this was not the case. The algorithm returned meaningless events such as [‘like’,‘not’]. Upon inspection, the 3 key words we were able to detect earlier have very low scores. However even injecting more tweets did not have a significant impact on the results. Doubling the injected tweets doubles the S_f score, however the score of words like 'manchester' (4.588) is still orders of magnitudes less than words like 'like'(3345.649). It should be noted that stopwords such as 'the' and 'to' have scores greater than 19,000. While these scores are out of place, another interesting observation is that all of these words are classified as 'aperiodic' i.e the dominant period P_f is equal to the entire time period we are looking for events in.

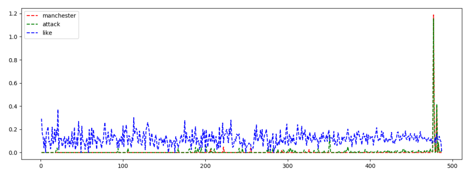

Figure 5: Feature Trajectory of keywords across time

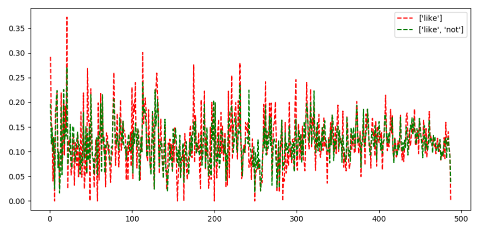

Figure 6: Visualization of events detected in a cleaned version of the MIBSample dataset across time

One of the disadvantages of using Feature-Trajectories is that it cannot completely be used in real time. This is because it builds a feature trajectory across time, and this feature trajectory changes every time new tweets are added to it. If a real-time system were to be built however, the document-frequency counter used in data representation could be extended after every epoch, however the rest of the algorithm would have to be recomputed every time, making it fairly impractical for large datasets that are frequently changing.

5.2 SigniTrend

It was clearly observed that SigniTrend is fairly sensitive to changing trends in the data set. Below is an ideal case example collected on a dataset that had sparse mentions of the words 'manchester' and 'attack' except for a small period of time when they occur densely. As the bucket with tweets with these keywords emerge, the significance score shoots up, and they are reported as trending words. Another interesting case was with the word 'uk' it occurs fairly often through out the dataset, with a gradually increasing frequency. So while it doesn't reach the same peaks as 'attack' it is generally trending throughout.

One can notice that when the Manchester-

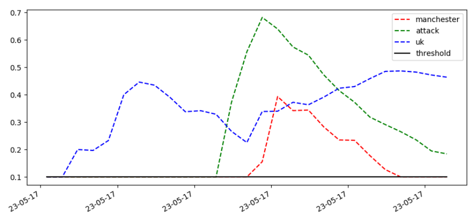

Figure 7: Tracking significant trends on cleaned version of the ManchesterAttack dataset with alert threshold of 0.1

Attack dataset is injected into MIBSample, upon detecting trends in the last 1 month (intersection between datasets) words like 'uk' are trending less prominently, whereas the spikes of words like 'manchester' remains.

One of the positives of using SigniTrend

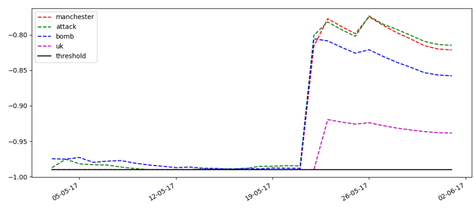

Figure 8: Tracking keywords for 1 month on MIBSample with ManchesterAttack injected

is that it outputs results in (almost)real time as tweets are being entered into it. By defining varying the length of the epoch, one can control how often tweets are processed by SigniTrend, and at the end of every epoch it will output which are the new trending topics.

Issues with SigniTrend emerged when the dataset was much bigger and noisier. The test below uses data from MIB sample, and is injected by ManchesterAttack. Ideally SigniTrend should detect the same keywords ('manchester' and 'attack') towards the end. However while it does detect an increase in the significance of these words, it requires a much bigger injection of tweets to generate a large enough significance score. Furthermore by lowering the threshold so that these words are trending, there are alot more words that emerge as trending that do not represent any event such as 'love' and 'fashion'.

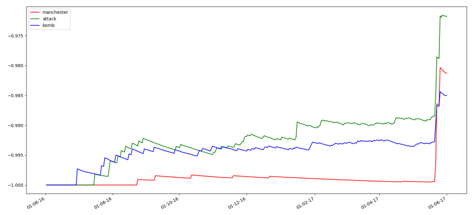

Figure 9: Tracking keywords for 1 year on MIBSample with ManchesterAttack injected

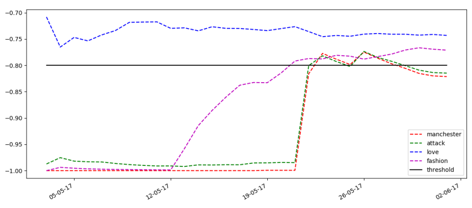

Figure 10: Noisy data on MIBSample with ManchesterAttack injected (1 month of data)

When it came to testing SigniTrend with datasets with more than one event, results showed the importance of not just the significance threshold, but also of when the attack occurs.

It can be observed that only the attack in

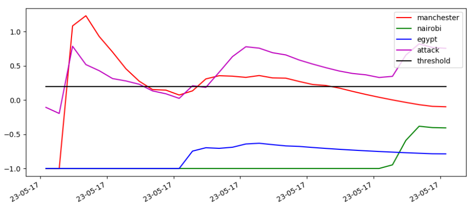

Figure 11: Tracking significant trends on the 3Attack dataset with alert threshold of 0.1

Manchester is reported, and not the Nairobi or Egypt attacks, even though the Nairobi attack had more tweets about it. Testing reveals that SigniTrend is more sensitive to bursts in the early cycles, but as the hashtable grows and has more data to compare incoming tweets to, it needs a greater number of tweets to detect the trend. It can also be noted that even though the exact attack event is not reported, after the Egypt attack the word 'attack' is reported even though the location is not. Similarly there is a spike in the score of 'attack' after the Nairobi attack.

5.3 EDCoW

EDCoW certainly is the most computationally intense technique out of all that we implemented. This had some serious flaws with regards to time-consumption. Unfortunately it was not possible for certain parts of the algorithm to be executed, and therefore had to be modified:

- Newman's modularity based graph partitioning algorithm was far too computationally intensive: In most tests attempted on the smallest dataset (ManchesterAttack), the code would timeout, and not return any events. This lead to us testing with other forms of graph partitioning algorithms. A version of K-means was adapted, but it had a terrible precision as it returned almost every possible cluster with the filtered words. Another attempt was made by using a greedy graph partitioning algorithm. This did have fewer events returned, but each event was described by way more words that was necessary, making it very imprecise.

- Epsilon value cannot be computed: In many cases the size of the cluster returned was far too large for epsilon (the minimum significance score of an event) to be computed in Python. This is because the exponential term of the calculation was too large to be stored in a Python variable. To solve this, a maximum cluster size was added.

In addition to these problems detected on a small dataset, it was not possible to test the algorithm on a larger dataset (MIB Sample), as the execution took over 8 hours, after which it was terminated by the kernel possibly due to resource starvation. Therefore its results are not included here.

Clearly a disadvantage of using EDCoW is that it cannot be used in real time. As a very computationally intense technique it would not be feasible to attempt to detect events in real time using EDCoW as after adding every tweet, the entire pipeline would have to be re-executed. This is certainly a problem given the huge volume of tweets coming every second, and will keep growing as more words are added to the bag of words.

5.4 Near Real-time Burst Topic Detection

While technique uses a rather simple, easy to implement algorithm, in terms of its results it was fairly inconsistent. Several issues came up including:

- The results are highly dependant on the parameters set, this is not ideal as it is not feasible to anticipate incoming data and tuning parameters accordingly. Testing with several different parameters lead to inconclusive results, however we did verify that setting a high similarityThreshold led to far more topics gathered in buckets, and a low similarityThreshold lead to very few topics in buckets. The values specified in the paper (0.5-0.6) seemed to work well.
- Results in the first 50-100 tweets were quite inaccurate, and most of the false-positives were returned right in the beginning.
- Doesn't scale very well, and for a large dataset it would be ideal to implement the cloud based processing using some technology like Trident API as described in the paper.

This algorithm as the title suggests does work really well on data being processed in real time, as it doesn't have to recompute from scratch whenever new tweets are added. Scaling issues aside, the algorithm can detect if a burst has occurred after every tweet that it processes, provided the right parameters are set.

6 Conclusion

There are many positives to take from this study. By implementing 4 different approaches to the topic of event detection, we were able to get a holistic understanding of the state of the art in the field. In particular, we were able to see that both SigniTrend and FeatureTrajectories provided promising results, especially on smaller datasets. In order to see better results on large datasets, the burst size must too be much bigger to "emerge" above the unimportant tweets. On the other hand, Real Time burst detection provided mixed results. While in theory the idea is great, the high variability of results based on parameters set is something too keep in mind.

On the other hand there were some negatives too, as several things did not work.

Primarily among these was not being able to correctly cluster signals into events in EDCoW. This can be attributed to using the wrong kind of partitioning techniques, but it wasn't possible to verify whether using Newman's method as described in the paper would have worked as the algorithm was too computationally intense to perform. Another issue that arose was the automatic threshold being set too high in FeatureTrajectories. This could be due to the fact that the technique was designed as a general event detection algorithm, and it may not have accounted for the high amount of noise from stopwords in social media. While this was handled for the purposes of testing by manually overriding the variable, it is not an ideal approach. A potential solution that could be considered this could be scaling down the threshold based on the distribution of term frequencies between stopwords and non-stopwords.

There is certainly scope for further work that can be done on these algorithms. Some ways that could help improve results would be:

- Further investigating the the relation between the volume of tweets in a burst event, and the significance scores computed by SigniTrend
- Figuring out a way to implement a better graph partitioning technique for EDCoW.
- Reimplementing RealTime burst detection using cloud based processing (like Trident API).

References

[1] Mib datasets. http://mib.projects.iit.cnr.it/dataset.html. Accessed: 2016-05-14.

[2] Twitter usage statistics. http://www.internetlivestats.com/twitter-statistics/. Accessed: 2016-06-06.

[3] S. Girtelschmida, A. Salfinger, B. Prll, W. Retschitzegger, and W. Schwinger. Near Real-time Detection of Crisis Situations. Technical report, Inst. for Application Oriented Knowledge Processing, Dept. of Cooperative Information Systems, Johannes Kepler University Linz, May 2016.

[4] Q. He, K. Chang, and E.-P. Lim. Analyzing Feature Trajectories for Event Detection. Technical report, School of Computer Engineering, Nanyang Technological University, July 2007.

[5] S. Petrovic, M. Osborne, and V. Lavrenko. Streaming First Story Detection with application to Twitter. Technical report, School of Informatics, University of Edinburgh, June 2010.

[6] E. Schubert, M. Weiler, and H.-P. Kriegel. SigniTrend: Scalable Detection of Emerging Topics in Textual Streams by Hashed Significance Thresholds. Technical report, Ludwig-Maximilians Universitt Mnchen, Aug. 2014.

[7] J. Weng, Y. Yao, E. Leonardi, and B.-S. Lee. Event Detection in Twitter. Technical report, HP Laboratories, July 2011.