PRIM Project Report

Text Mining on Twitter

--Company event detection

FENG Yao && GUO Lieqiang

Supervised by prof. Mauro Sozio

Postdoc. Oana Balalau

29/01/2018

Table of contents

Abstract

The aim of the work is to detect interesting IT company events in the context of social media, Twitter. After gaining insight on existing event detection techniques, we implement a γ-quasi clique based method. Our experimental evaluation is based on a collection of tweets, showing a good performance about company events in social media.

Keywords

Event detection; social media; graph mining; quasi clique

1 Introduction

Text mining is an analytical method to derive relative information from text data. Text mining is getting a lot attention these last years, due to an exponential increase in digital text data from web pages and social media services.

Online social media is a rich information source and Twitter, as a social media, have more than 300 million active users each month and about 500 million tweets posted per day, which provide us a good insight about the offline life. Event detection has been focusing on traditional news media for a long time, but online social media also provides a good source for data mining and social network analysis. Event detection on social media presents huge challenges but also unprecedented opportunities. The difficulties of text mining on twitter are that tweets are written in informal language. Besides, tweets may contain typos and abbreviations. In most time, useful information intertwines with noises such as spams and posts by twitter bots.

On the other hand, tweets might provide novel information about well-known events or even about events that have not been covered at all or quickly dismissed by mainstream media. Twitter data constitutes a rich source that can be used for capturing information about any topic imaginable. For example, users may complain about the newly published electronic products, which will be noticed by the main media only when the problem become serious and many users are affected. With text mining on Twitter, we are able to identify these potential problems at the first beginning. Those tweets which make comments on companies' services and products are what we focus on. It could help company to find possible problems existing in their products and it is a good way to get direct feedback from users.

1.1 Problem definition

We focus on company events, especially for IT company. The company events are defined as positive comments or negatives comments posted by a large group of people, mainly the clients of the company.

Given a set of tweets, we try to find a number of events about a specified company by filtering an entity. Due to the time limitation, we only built a dataset around 2 months, which allows us to obtain meaningful insights on the important event.

2 Dataset

In this section, we describe the input datasets for experiments. Twitter Streaming API is an official Application Programming Interface that we can makes the interaction with computer programs and web services in real time. For example, we could listen for tweets that contain specific strings or actions a user might take.

2.1 Lexicon

Twitter Streaming API allow us to download tweets satisfying several constraints. In this project. We built a lexicon which is a list of keywords. The downloaded tweets must contain one of these keywords to reduce the range.

It is not easy to build a IT company lexicon. We have reviewed some papers about crisis event detection, of which the lexicon is easier to find, such as flood, earthquake.[1][3] But for our case, we have no reference about how to identify a company event. One idea is to begin from the products and problems.

In our lexicon, there are common electrical problems such as explode, crash, broken, etc. They may be related to phone explodes, system crash, screen broken and other possible problems. There are also important parts of electrical products such as screen, camera, battery, which may be related to user comments.

It is obvious that these keywords may also refer other events, such as relationship issue due to the word ‘broken’. We’ll use an entity to filter tweets. For example, we only focus on tweets containing the word ‘Samsung’ to find events about Samsung company.

2.2 API limits

Twitter Streaming API has several limitations for accessing data in real time.

Firstly, tweets can be queried by the contents. However, the contents do not mean the content of tweet, but the content of a Json element. For example, if we want to download tweets containing the word ‘iPhone’, we may obtain tweets that is posted via an iPhone and the contents have nothing to do with ‘iPhone’. As we try to detect IT company events, this causes huge problems, which are solved by deleting all tweets without a key word existing in the contents.

Besides, independently of the method used to query, the resulting set is limited to 1% of the stream data. If the query matches more than 1% of the data, then the data is subsampled uniformly at random. Even though we use a blank lexicon, we cannot get more than 1% data. As we generalize our lexicon to widen the range, we may lose tweets due to this limitation. But the most important events will never lose because there are many tweets referring it.

2.3 Datasets

We get a dataset of 55,034,364 tweets of 65 days between November 2017 and January 2018. All tweets collected are written in English which may come from Europe, America, Australia and other English-speaking areas. On average, we have 856 k tweets every day and data are separated by the dates. Thus, it is possible to pick several days for the experiment.

When listening to tweets in real time, we delete tweets of which the keywords exit in other keys such as rather than text. Here is an example: "rel=Twitter for iPhone". Therefore,

in our results data, all tweets contain at least one word in lexicon in the key "text". Besides, we only keep "created_at" and "text" in the json file to reduce the disk space and accelerate the reading process. Dataset is available from [4].

3 Model

Our model consists of the following main steps: 1) collection of tweets containing keywords related to crisis events by means of the Twitter API; 2) data preprocessing; 3) burst detection according to tweets frequency; 4) event detection in given bursts.

3.1 Tweets collection

Tweets have been collected by means of the Twitter Streaming API with our lexicon introduced in the section 2.

We have tried other methods to build our lexicon. One is to use company names such as Nokia, Samsung. But it lacks the generalization and it is limited to these predefined companies. If we want to find interesting events about other companies, they must have a relationship with the predefined one. Another is to use strict keywords such as screen broken, glass shell. Although the range of companies becomes larger, the events are limited. We could not find other possible events about screen and glass, because it is not possible to predict all user's feedbacks. That's why we try to build a generalized lexicon which allows to liberate any possible restrictions on tweets.

3.2 Data preprocessing

The choice of our lexicon has a huge disadvantage: there are many useless tweets downloaded compared with the crisis events detection lexicon. In data preprocessing, we'll given an entity and filtering tweets containing it. It guarantees that the remaining tweets are at least related to the entity provided. For example, the keyword 'explode' may refer to terrorist attacks which has nothing to do with our work. But after filtering with an entity 'Apple' or 'Samsung', all these tweets lacking the entity will be removed.

When preprocessing the tweets, they are transformed in lower case. 'SAMSUNG' and 'Samsung' will become lower case, 'samsung', and we'll not miss any information due to the case confusion.

We'll remove links, special characters and emojis in the tweets. For emojis, we keep words only consisting of digits and letters rather than using regular expression. The regular expression cannot completely remove all emojis because there are always new ones created by netizens.

They the tweets are tokenized to several words after removing stop words and they are stored in a list to represent the original tweets. The operation tokenization is to automatically separate words by the space. Besides, the corresponding published time are also recorded in order to mark tweets in timelines. The following steps are based on the tokenized words.

3.3 Burst detection

When a hot topic occurs, we observe a burst of activity in Twitter with terms pertinent to the event increasing suddenly their frequency in tweets. Although there exist good events which

are not largely referred by others, the burst represents an important event with high probability.

Given a window size l, we could calculate the number of tweets in every period, defined as N(i) where i is the index of period. If the total number of tweets in our dataset is NUM, then we could get the number of period L = $\left\lceil \frac{NUM}{l} \right\rceil$. The burst detection is simply according to the frequency N(i) and we pick the largest several frequencies as the burst. Besides, it is possible to integrate the neighbor period into the same burst because the length of burst is often variant and it could reduce the possibility of event overlap. The adjacent bursts tend to indicate the same of similar events. Algorithm 1 shows a pseudo code for computing all bursts in our collection of tweets.

Algorithm 1 Burst detection

Input: A collection of tweets C with their timestamps, a window size l and the thresholds α

Output: Burst set S and a counter of words W

```
Derive the number of tweets N in every window
S = Ø, W = Ø, NUM = N(C), L = ⌈NUM/l⌉        #initialization
For i = 2 … L-1                    # traverse all period except the beginning and the end
    If N(i) >α && N(i) is a local maximum
        x = {i}, y = word frequency in x
        For j next to i, if |N(j) – N(i)| < β     #include neighbor periods into same burst
            x = x ∪{j}, y = word frequency in x
        End
        S = S∪x and W = W∪y                #update S and W
    End
End
Return W and S
```

Here is an example of burst detected for entity 'Samsung' for 30 days data (11th November 2017 to 10th December 2017) shown in figure 1. The window size is 4 hours, the threshold α is set to be the mean value of frequency, the threshold β is set to be the 5% of maximum difference of frequency and we pick the 10 largest bursts. We can see that bursts N.3 and N7 contain more than one period due to a close frequency.

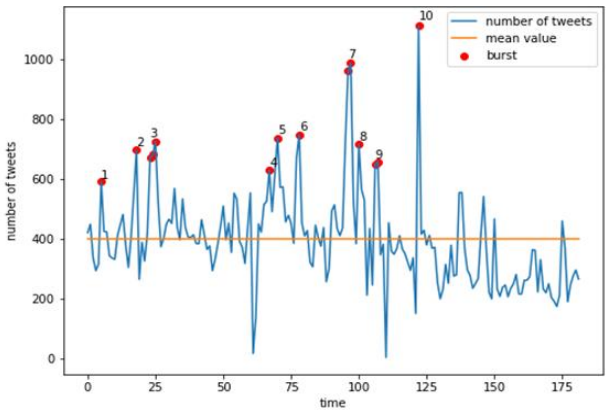

Figure 1: 10 largest bursts for entity 'Samsung'

3.4 Event detection

3.4.1 Co-occurrence matrix

We then focus on the resulting bursts. We get could calculate the term-document matrix M for all words appearing in the burst. It is also possible to build a term-document matrix for N most important words in the burst according to the words counter in algorithm 1.

The term-document matrix is a mathematical matrix that describes the frequency of terms that occur in a collection of documents. The term is the word and the document is the collection of tweets. Therefore, the term-document matrix in our project measures if a word appears in a tweet. It is noted by M(i, j) = 1, if word i exits in tweet j. If there are N words and NUM tweets, then M has N rows and NUM columns.

The co-occurrence matrix C measures how many times two words appear in the same tweet. It is noted by C(i, j) = w, if word i and word j appear w times in the same tweet. In another word, there are w tweets containing both word i and word j. C is a square matrix of size (N, N). It is calculated by $C = M^T M$.

3.4.2 Quasi-clique detection

3.4.2.1 Quasi-cliques

Obtaining the co-occurrence matrix for each burst, now we are going to find out interesting events. We could get a weighted undirected graph G from the co-occurrence matrix. The events are related to a given company, and we denote it as node v.

To get a description of an event, we want to find a set of terms all related to the same event, which must be strongly connected with each other. Thus, these terms induce a dense region in the graph. Now we need to find the densest region.

Cliques aim to find dense subgraph. There are several definitions of weighted cliques, such as the most weighted subgraph with limited nodes. If we use this definition, we may find a subgraph like the left one. The left one doesn't satisfy our requirement, because there are strong connections between the set of nodes of {1, 2} and {3, 4}, while other connections are much weaker. However, in the right graph, all nodes are connected with same weight, showing an equal contribution to the event.

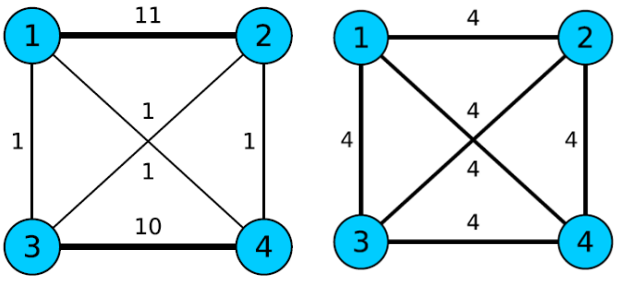

Figure 2: quasi-clique examples. The total weight of the left graph is bigger than that of the right one, while in the right one, nodes are better connected.

The graphs similar to the right one in Figure 2 are the interesting cliques. However, we may not be able to find this kind of perfect cliques with equal weight. We want to find a subgraph with similar weights among nodes. This motivates the definition of γ weighted quasi-clique:

Definition (γ-quasi-clique) Given an unweighted graph G = (V, E, W), and 0 < γ ≤ 1, G is a γ weighted quasi-clique if and only if the following inequality is satisfied:

$$\sum_{e \in E(G)} w(e) \geq \gamma \bullet w_{max}(G) \binom{|V(G)|}{2} \quad (1)$$

where w(e) denotes the weight of edge e, $w_{max}(G)$ denotes the weight of the most weighted edge in G.

3.4.2.2 Algorithm

In a quasi-clique, all edges have similar weights. Finding a quasi-clique is a problem of NP hard, but in our case, the complexity can be reduced by restrict the quasi-clique to begin from a give node v and at most MaxN nodes. [1] The node v is the entity used to filter tweets, which is normally the name of a company. Algorithm 2 shows a pseudo code for finding the γ-quasi-clique in a given graph G.

Algorithm 2 Find γ-quasi-clique

Input: A graph G = (V, E, W), a vertex v, an integer MaxN, 0 < γ ≤ 1, a threshold α ≥ 0
Output: A weighted quasi-clique S containing v

```
u = argmax_{u :uv∈E} w(uv)        # w(uv) means the weight of edge between u and v
x = argmax_{x≠u :xv∈E} w(xv)      # x is the node with second largest weight
While w(uv) > α•w(xv) do          # avoid weight to be too large
    V = V \ {u}                   # delete u from node set V
    u = argmax_{u : uv∈E} w(uv)   # recalculate u and v
    x = argmax_{x≠u : xv∈E} w(xv)
End
S = {u, v}
While |S| ≤ MaxN do               # traverse all nodes until S contains MaxN nodes
    Δ(S) = ∪_{u ∈S} Δ(u) \ S
    If Δ(S) = ∅
        Return S
    End
    x = arg max_{u ∈Δ(S)} q(S ∪ {u})
    If q(S ∪ {x}) < γ
        Return S
    End
    S = S ∪ {x}
End
Return S
```

In algorithm 2, the inputs include a weighted graph of which the adjacent matrix is our co-occurrence matrix C. γ is defined as the similarity of the average weight and the maximum weight in the subgraph. The quasi-clique is initialized as a set only containing the node v, which is the entity (company name). The general idea is to find most weighted nodes satisfying the equation (1).

Here is the algorithm described in [1] to find the quasi-clique. We start with v and find next node u which has the most weighted edge with v. Add u and v to the clique set S. Next, we find all the neighbor nodes which does not belong to S for each node in S and put one of them to a set Δ(S); if Δ(S) is empty, we are done by returning S. Else, we are going to find new node x in Δ(S) which satisfies equation (1) and maximize q, which is defined as:

$$q = \frac{\sum_{e \in E(S)} w(e)}{w_{max}(S) \binom{|V(S)|}{2}}$$

We repeat the loop until the number of nodes in S is enough or no more nodes could satisfy the equation (1).

Note that our algorithm 2 is a little different from the one above one. For the first node besides v, we'll examine if it is much larger than the second weighted node. Otherwise, we may be restricted by the large weight and cannot find a good quasi-clique of which the maximum weight is smaller. This modification is due to experimental results. For example, if we use 'Apple' as the entity, 'iPhone' is always in the resulting quasi-cliques as the most weighted nodes. As they always co-occur, the weight is much larger than any other nodes but it is meaningless to have it in our cliques. In addition, many useful words may be ignored for not satisfying the equation (1). We can also remove the improper word from our lexicon if it is only connected to few nodes in the graph.

We could use the resulting quasi-clique words to describe the possible events. But after removing stop words and hashtags, they may be hard to explain the event. We try to find a tweet which matches most of the words in quasi-cliques to give a description of the events. This could help us well understand what the event means.

Here is an example showing the modification of algorithm 2. The figure 3 shows an example graph G where there exist two (quasi) cliques. The right one is a quasi-clique of three words and the weights are large. The left one is a quasi-clique of five words and the weights are smaller. Obviously, the left one is a good event while the right one means nothing. Without of verification of the first node, 'iPhone' will be added in the clique as it is the most weighted node for root word 'apple'.

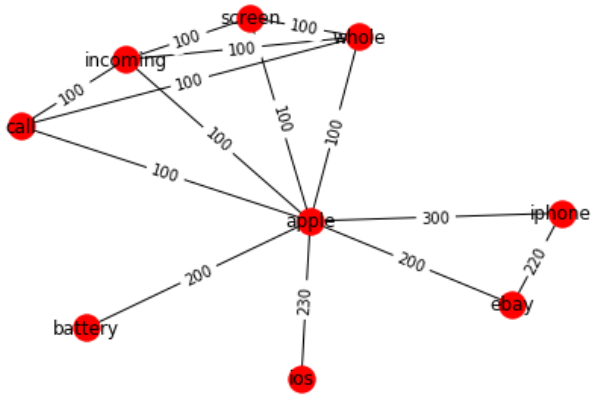

Figure 3: An example graph G for identifying the cliques

According to algorithm 2, 'iphone', 'ebay', 'ios', 'battery' will be removed one by one and finally we get the left quasi-clique as the result. The improvement of the modification of algorithm 2 is shown in Table 1.

Table 1 the modification of algorithm 2

No modification	u'apple', u'iphone', u'ebay', u'screen', u'take', u'amp', u'ios', u'battery', u'update'
Modified algorithm	u'apple', u'screen', u'come', u'whole', u'comes', u'call', u'incoming'
Tweet	RT @unapologetiicb: Apple needs to come up with a feature that when an incoming call comes in it doesn't take up the whole screen so yo…

The first two rows of Table 1 present the clique words and the last row shows the corresponding tweet. It is obvious that the modification really improves the performance for detecting events. Besides, the word "doesn't" is an important word in this event but it is removed as the stop words. That's another reason why we need find the tweet to better explain the event.

3.5 Update lexicon

In collecting tweets, the lexicon plays a very important role. As we described in section 2.1, our lexicon is not a mature system. We want to improve it by finding more candidates of words that describe the feelings while using some companies' products

The basic idea for expanding lexicon is that the words co-occur with our lexicon may also describe problems in the same domain. [2] We assume that if a word appears very frequently with these words in our lexicon, it has a high probability that it's a good candidate for expanding our lexicon. Algorithm 3 shows a pseudo code for performing page-rank.

Algorithm 3 page-rank (M, P, itr, idxs)

Input: matrix M with size N×N, parameter 0 < P ≤ 1, maximum iteration itr, index
Output: page-rank scores R

```
Define a constant ε
Do L1 normalization for each column of M
V1 and V2 are two vectors with size N×1, the initial value is 1 for all entries.
For i = 1, 2, …, itr
    V2 = (P*M+(1/N) *(1-P) *IN) *V1   # IN is a ones matrix with size N×N
    If L2 norm of (V1-V2) < ε
        break
    End
    V1 = V2
End
W = M • diag(V2)     # diag(V2) is a diagonal matrix transformed from V2
For i = 1, 2, …, N
    Ri = 0
    For j = 1, 2, …, n # n is the number of words in lexicon
        Ri=Ri + W [i, idxs[j]]
        # idxs[j] represents the row (or column) index of the j^th word in lexicon M
    End
    Ri =Ri/V2i # V2i is the ith entry of V2.
End
Return R                 # R = [R1, R2, …, RN]
```

We firstly construct a co-occurrence matrix M for a set of tweets. We think that $M_{i,,j}$ measures the degree of how related the word i and the word j are. Then We normalize M by column with L1 normalization, and next we perform page-rank algorithm on it and we can get a page-rank score for each word.

When we get the page-rank scores, we then sort them in descending order and output the corresponding words as candidates to a file. Then we can choose words from the file to expand our lexicon.

4 Experiment Results

4.1 Experimental settings

The dataset used for experiment is posted from 11th November 2017 to 14th January 2018. Using Twitter Streaming API, we make sure every tweet contains at least one word in our lexicon. Due to the polysemy, the tweets are not certainly related to company events. Then we use an entity to filter the dataset and the remaining tweets are all related to the entity. The entity is often chosen to be the company name or the popular products, e.g. Samsung, iPhone.

Our algorithms have been implemented in python and code for reproducing our results is available from [4]. There are some key parameters to be decided for the experiment. We set the window size l to 4 hours, which is a pretty small value because we could integrate neighbor periods to the same burst. A large window size may result in losing some events. For the quasi-clique parameter γ, it is set to 0.45. The quasi-clique parameter should not be too small to avoid unrelated words being included in our quasi-cliques. But if it is too large, we'll get the 'perfect' clique which may not exit in real data.

We'll find many bursts but every burst does not result in an event. Firstly, a burst of tweets can coincidentally happen instead of being caused by a hot topic. In this case, there are fewer words in our resulting cliques compared with a real event. Thus, a quasi-clique is not added in our final results if the number of words is not enough. Secondly, some bursts may lead to same event, sometimes even the same tweet retweeted by others. If two bursts induce the same tweet, they are overlapping and we'll ignore the latter one.

4.2 Evaluation

There are many advertisements of selling products in our results, which are widely retweeted. They are not events we are interested in. The company events include positive or negative comments about the company service of their products. Thus, we use the precision to describe how many positive/negative events are found in our results.

4.2.1 Performance

At the first beginning, we got a very bad result with many advertisements and only a small number of interesting events. This is due to improper keywords in our lexicon such as 'phone' and 'smartphone', which do not contribute to interesting events but advertisements.

The resulting events are shown in Table 2 and Table 3, respectively for the entity 'Apple' and 'Samsung'. Negative events are defined as complaints about the products or problems.

Positive events are defined as praises about products or recommendations. No events are defined as other possible results such as advertisements.

Table 2 : 20/40/60 bursts for Apple

N important bursts	20	40	60
Events	13	21	30
Negative events	6/13 = 0.461	9/21=0.428	12/30 = 0.40
Positive events	3/13 = 0.230	5/21=0.238	7/30 = 0.166
No events	4/13 = 0.307	6/21=0.285	11/30 = 0.366

Table 3 : 20/40/60 bursts for Samsung

N important burst	20	40	60
Events	15	25	37
Negative events	2/15 = 0.133	4/25=0.16	11/37 = 0.297
Positive events	4/15 = 0.267	6/25=0.24	6/37 = 0.162
No events	9/15 = 0.6	15/25=0.6	20/37 = 0.54

With same parameters, there are more useful events for Apple than Samsung. A precision between 60% - 70% is already not bad for entity Apple. However, the precision is less than 50% for entity Samsung, which is not ideal. Therefore, we have to do some changes in order to improve the performance on entity Samsung.

After observing the No events of Samsung, we find that advertisements are always followed by a link for selling phones, cases, etc. We remove all tweets containing links in the data preprocessing step and the results for entity Samsung are shown in Table 4.

Table 4 : Improved results for Samsung

N important burst	20	40	60
Events number	14	26	35
Negative events	5/14 = 0.357	8/26 = 0.307	13/35 = 0.371
Positive events	5/14 = 0.357	10/26 = 0.384	12/35 = 0.342
No events	4/14 = 0.285	8/26 = 0.307	9/35 = 0.257

As expected, the performance has been largely improved. The precision rises from less than 50% to around 70%. It proves that a tweet containing a link is more likely to be an advertisement instead of a meaningful event. Of course, we may lose some information by removing all tweets with links but the compromise is worth to do.

People tend to give negative comments rather than positive feedbacks in our results. This may be caused by our lexicon, which contains many potential common problems of electronical products.

4.2.2 Example events

The example events related to Apple and Samsung are shown in Table 5. The hot topics are included like the iPhone's battery issue. The tweets are that Apple slows down iPhones with old batteries to force you upgrade a new iPhone. Some of them are very interesting. For example, Samsung has the technology to explode since 1972, which is ironic.

Table 5 : Example events

Entity	Description	Examples
Apple	Face ID	Apple's Face ID security unlocked via a mask containing a silicone nose, 3D printed frame, and printed pictures…
	Security	That patch Apple rushed out for its "root" security flaw?
	Upgrade	anybody telling me to upgrade my phone is a slave to apple and capitalism.
	Battery	Apple quietly throttles your iPhone CPU as the battery gets older.
Samsung	Face ID	Who needs Face ID? Samsung wants phones to read palms
	Battery	Samsung Galaxy Note 8 users reportedly unable to recharge completely flat battery
	Technique	"Samsung Develops Battery Material with 5x Faster Charging Speed." Most of these battery 'breakthroughs' never made
	Explode	The new Samsung Galaxy Note 7 has a tendency to explode. Not to brag but we've had that technology since 1972.
	camera	Nice Samsung what camera do you think is better
Apple/Samsung	Ads	"Pokemon Pokeball wallet iphone case, samsung galaxy phone case Get it her https://t.co/Dt8Mhn3MxJ

4.2.3 Popularity degree

We could also give a rough description about the popularity of a company. All results are from the same dataset downloaded via the same lexicon. If a company has more tweets than others, we could think that it is more popular or that it is hot in this period. As there are many negative comments, we can say, it gets at least more attention than others companies with more tweets related.

The number of tweets filtered by the company name are shown in Table 6. It presents that Apple and Samsung are more likely to be discussed in social media.

Table 6 Number of tweets for different companies

Entity	Apple	Samsung	Huawei	Nokia	LG
Tweets	114,839	13,099	4257	2,476	2,179

5 Conclusion and Future Work

Motivated by large scale data produced from social media, especially Twitter, we study a methodology in a new domain of event detection: IT company event. Different form the crisis event detection, we build our generalized lexicon to enhance the range of collected tweets. We use an improved γ-quasi-clique method to identify the events in the bursts. The experimental evaluation shows the effectiveness of our model with a good precision. Our work displays that hot topics about companies are likely discussed in Twitter and the number of tweets could give a rough idea about the popularity of a company.

For future work, we would like to use a natural language processing method to automatically judge if an event is positive, negative or meaningless. Due to time limitation, the dataset is not large enough. We could apply our model on large scale data.

Acknowledgements. We thank our supervisor Mauro Sozio for detailed advice during the whole project. We would also like to show our gratitude for Oana Balalau for helping us a lot in the implementation part.

Reference

[1] Oana Denisa Balalau and Mauro Sozio. EVIDENSE : Allowing a Large-Scale Analysis of the Coverage of Crisis Events in Social Media
[2] Olteanu, Alexandra, et al. "Comparing events coverage in online news and social media: The case of climate change." Proceedings of the Ninth International AAAI Conference on Web and Social Media. No. EPFL-CONF-211214. 2015
[3] Olteanu, A.; Castillo, C.; Diaz, F.; and Vieweg, S. 2014. CrisisLex: A lexicon for collecting and filtering microblogged communications in crises. In Proc. of ICWSM.
[4] https://github.com/hgjt8989/PRIM