Event Detection

Mohammad Otoofi

2263537o

1 Introduction

In the domain of social networks analysis, like Twitter, exploiting and analyzing data generated by users can give valuable information and insight about public opinion regarding an issue, detecting an event, or analyzing business information. As mentioned, one way of exploiting twitter data is to detect events based on the information being reported by users. However, there are two problems which need to be addressed regarding to event detection. First, the volume of data is being generated by users is so high, thus, an algorithm is needed to scale up with data. Second, to relate twitts to an event, they need to be classified into clusters. Since, many of the clusters could be noise and talking about trivial issues, they can be put under more constraints to find real important clusters .

In this coursework, the latter problem, improving precision of clustering is addressed and for this purpose, burst detection approach is used to detect events. The method used for burst detection is based on the approach described in McMinn and Jose (2015). To implement burst detection, it is needed to have streaming twitts. Particularly, to create sliding windows for burst detection, the data is needed to be sorted by time. Thus, The clustered data called "clusters.sortedby.time.csv" is exploited. In this way, sliding windows can be created based on the twitts' time stamp.

Moreover, the data has been already pre-processed and filtered. Each twitt has a cluster name entity which describes the twitt topic. Each twitt is also grouped in a cluster which has an unique id. In this coursework, it has been tried to cluster the twitts again based on bursting approach. Thus, the current clusters and their id are ignored. However, cluster name entity is still used to create inverted index matrix.

1.1 Burst detection with sliding windows

Burst detection approach is based on another method called sliding windows. Basically, in sliding windows, which start with 5 minutes length and doubled for the next window, a start and end time are determined and then twitts, covered by that window, are counted. In this coursework, different number of windows are tested and their effect on clustering precision is showed.

To implement burst detection, twitts in "clusters.sortedby.time.csv", are read one by one. The first twitt time stamp is flagged as the beginning of streaming i.e. window start time. Each time, a new twitt is read its time stamp is compared with the window start time. If it is still covered by the length of the window then it is added to the inverted indexes. Otherwise, the windows will be checked for any bursting entity and also, the start time of the window is updated to the last twitt time stamp. In this way, the window moves over new twitts. This part is shown in appendix A algorithm 1.

To check whether an entity is bursting, its frequency in the window is compared with a threshold which is computed with the mean and standard deviation of the window that is $3 * \sigma + \mu$. In other words, each window has mean and standard deviation which are updated periodically according to the length of the window. Detecting bursting entities algorithm is shown in algorithm 2.

To compute mean and standard deviation of a window more efficiently, three sigma rule is applied. Regarding to the volume of streaming twitts in the real time, three sigma rule helps to compute mean and standard deviation of the windows with less computation. Based on three sigma rule, three numbers are maintained for each window and they are updated every time a window is finished. These three numbers are called S_0, S_1, and S_2 and computed as follows:

$$S_{jn+1} = S_{jn} + S^j_{n+1} \tag{1}$$

Now, based on these values, mean and standard deviation can be computed:

$$\mu = \frac{S_1}{S_0} \tag{2}$$

$$\sigma = \sqrt{\frac{S_2 - S_1^2}{S_0}} \tag{3}$$

The pseudocode of three sigma rule and how windows get updated is shown in algorithm 3.

Finally, if an entity frequency in inverted indexes is greater than the window threshold then it is considered as a bursting entity or an event.

1.2 Traffic event detection

To tackle problem of traffic event detection, I propose a method based on classifiers. Classifiers can be used in two ways either, after clustering or before clustering twitts. More specifically, in the first method, having clustered twitts or events, using classifiers, it is tried to detect whether a cluster or event is related to traffic. In the latter method, when a twitt is posted it is decided whether the twitt is related to traffic. Then, if the number of related-traffic twitts in a specific time interval is greater than a threshold it is considered as a traffic event.

However, regarding the high volume of data, which is being generated by users in real time, classifying each twitt after being posted is not efficient. Particularly, when an event happens even the volume of data raises to a level higher than ordinary times. Thus, it is difficult to classify twitts in real time one by one.

For this reason, I turn to the second solution which is to classify clustered twitts. In fact, instead of classifying twitts one by one, we try to classify clusters, or events, into two groups, traffic events and non-traffic events.

One of the-state-of-art classifiers is neural networks. Using regresion, currently, neural nets have shown good performance in image classification and sentence classification (LeCun et al. (2015)). However, to use any kind of classifiers, first, features in data need to be specified. For traffic event detection, a set of keywords related to traffic can be considered and then extracted from twitts' text. For instance, "crash", "overload", "traffic", "accidents"," heavy", and "congestion" can be used as the keywords. Each time a twitt is posted it is processed to create one-hot-vector based on the keywords. Having created one-hot-vector for a twitt it can be used as an input to a neural network to classify it as a traffic-related or non traffic-related twitt. More precisely, when a cluster or event is detected its twitts is used to be fed to neural net. To make it efficient, twitts in a cluster is used as a batch and stochastic gradient descent is considered as the neural net optimizer. The output, then, is probability of being related to traffic for that event. As shown in equation 4, neural networks uses simple regresion for classification task. In our case, X is the on-hot-vector and y is the probability of being related to traffic. For more clarification, if the set defined above is considered as the keywords and there are 40 thousands twitts in a cluster then the dimensions of the X would be 40000×6. The algorithm 4 shows a pseudocode for traffic event detection.

$$y = WX^T + b \tag{4}$$

However, the problem with all the classifier methods, not only neural nets, is that they need training data set and it needs to be annotated. In traffic event detection, it is needed that clusters or events are assigned either true or false label. In addition, selecting a set of keywords, which can reflect truly the set of words used frequently by people for traffic, is crucial.

2 Code description

2.1 Burst detection with sliding windows

This algorithm is implemented by python and only one class is used for its implementation. This class has five attributes and methods. Its class diagram is shown in the figure 1.

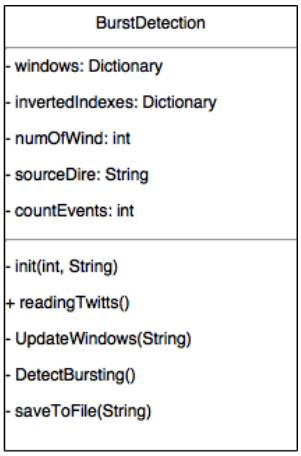

Figure 1: Burst detection class diagram

To implement burst detection, two dictionaries for inverted indexes and windows are used. The first data structure is a dictionary called "windows" which holds information about entities and three sigma rule parameters. The second data structure is another dictionary to hold inverted indexes.

"Windows" dictionary is a three dimensional dictionary. In other words, it is a three dimensional matrix which its rows is entities, its columns is windows, and its depth is S_0, S_1, and S_2 to compute windows mean and standard deviation.

The second dictionary, called "invertedIndexes", is to hold entities and twitts correspond to them. As explained before, in the pre-proccessing stage, a cluster name entity is extracted for every twitt. Cluster name entity is an expression consisted of one or more words and explaining what the topic of the twitt is about. It is used as the key of "invertedIndexes" dictionary. The value of each key is a list of twitts with the same cluster name entity.

As shown in the algorithm 1, burst detection starts with reading twitts one by one. If the cluster name entity of the twitt has already added to the "invertedIndexes" then the information of that twitt, like twitt id, user id, and text, will be added to the list of the twitts related to that entity. Otherwise, a new element will be created for that cluster name entity. Then, all the windows are checked to see whether a window is over. If a window has been over then first, entities in "invertedIndexes" are checked for bursting. Second, the window's parameters are updated and finally, if it is the largest window the inverted indexes is set to empty. This procedure is implemented in readingTwitts().

To detect bursting entities, frequency of each entity in "invertedIndexes" is compared with the window threshold, $3\sigma + \mu$, and minimum number of entity frequency, 10. As it is shown in algorithm 2 and implemented in DetectBursting(), σ and μ is computed based on three sigma rule with S_0, S_1, and S_2.

In the final step, the parameters of the window, which is over, are updated based on the current frequency of entities in "invertedIndexes". This is shown in algorithm 3 and has been implemented in UpdateWindows(windId).

These pseudocodes are implmented in a script called EventDetection.py. The script EventDetection.py gets an argument for the number of window. For instance, to run the script with 2 windows, you can use command below:

```
python EventDetection.py 2
```

In addition, the scripts requires to be put in the data folder otherwise the directory of input file can be used as the second argument to the script.

```
python EventDetection.py 2 DATA_DIRECTORY/clusters.sortedby.time.csv
```

2.2 Traffic event detection

Traffic event detection is not implemented in this coursework, however, a high level possible implementation of it is proposed.

To Implement traffic event detection, two dictionaries, ,called "invertedIndexes" and "windows", are used. However, "invertedIndexes" usage is different from previous method. In addition to holding frequencies, "invertedIndexes" is to keep one-hot-vector of twitts. Every time a window is over bursting entities in that window are detected based on three sigma rule. Then, the twitts correspond to the bursting entities are extracted from "invertedIndexes", which also keeps one-hot-vectors. Further, these one-hot-vectors are used to be fed to neural network for classification. The output which is the probability of being related to traffic event is compared with a threshold. At the end, the window parameters are updated and if the window is the largest window "invertedIndexes" will be set to empty. A high level pseudocode is show in appendix A algorithm 4.

3 Evaluation

To evaluate burst detection, we need some measures. To show the accuracy of the clusters, cluster precision is used. It is calculated from number of the detected clusters i.e. bursted entities along with number of matched back clusters. Also, number of detected events is reported. It is used to calculate event recall. The higher these values, event recall and cluster precision, a better algorithm we have. Finally, form event recall and cluster precision, F-measure is calculated which is to show trade-off between these two values. However, F-measure has a drwaback. For instance, we can have a algorithm that has a high precision but its event recall is low. In this case, F-measure is still high. We will see such a result in the following paragraphs.

First, the effect of number of windows on F-measure is reviewed. The results are summarized in the table 1. The first method, which has only one window, has reached F-measure equal to the baseline. However, with a closer look, it can be seen that the number of detected events for one window is only 7 while the baseline is 106. The reason why F-measure is high is that the first experiment only has detected 37 clusters from which 11 of them are matched back to an event. This has caused a high cluster precision and subsequently high F-measure, while, one window's event recall is much less than the baseline. From this comparison, it can be inferred that just observing F-measure is not enough and it is needed to improve all measures. For this, the experiment continued into seven windows.

By increasing number of the windows just to two, a significant improvement can be observed in all the measures except the cluster precision which has a very slight decrease. This can be explained through the increase in the number of the clusters. Even though, both, detected events and matched back clusters, increased, but because lots of detected clusters are noise and background topics cluster precision decreased. However, the overall improvement is positive and convince me to continue the experiments.

This improvement was continued until the experiment with seven windows. In this experiment, because of the large number of the detected clusters, F-measure decreased. Despite of increasing in event recall, cluster precision decreased from the previous experiment. which was 0.269, to 0.249. However, even with decreasing in F-measure, number of matched back clusters and detected events increased from 13591 to 27367, and 101 to 105, respectively.

In addition to 7 windows suggested in McMinn and Jose (2015), one more window is tested. However, by increasing number of windows to eight, F-measure decreased significantly from 0.226 for 7 windows to 0.217 for 8 windows. Although, event recall and detected events are at the highest level with eight windows but cluster precision is even smaller than the experiment with 5 windows.

In addition, the results for 1 day dataset is shown in appendix B table 2.

Number of Windows	Event Recall	Cluster Precision	F-Measure	Clusters	Matched Back	Detected Events
1	0.014	0.297	0.026	37	11	7
2	0.101	0.277	0.148	1204	333	51
3	0.138	0.302	0.190	3929	1186	70
4	0.166	0.289	0.211	9391	2713	84
5	0.182	0.285	0.222	22003	6263	92
6	0.200	0.269	0.229	50551	13591	101
7	0.208	0.249	0.226	110025	27367	105
8	0.209	0.224	0.217	220487	49475	106
Baseline	0.209	0.014	0.026	50631	696	106

Table 1: Burst detection with sliding windows on 7 days dataset

4 Discussion

The best F-measure is obtained with six windows. However, seven windows has better event recall and detected events. It can be said that in terms of a trade-off between F-measure and detected events having seven windows is preferable. Although, from the number of the clusters not matched back, it can be concluded that using only a burst detection strategy is not enough to achieve a better cluster precision.

To make cluster precision more accurate, a possible extension is using a text classifier. Text classifiers can be used to improve cluster precision. I mention some of the previous works based on text classifiers. These methods could be used for both, increasing cluster precision and to classify events into traffic-related events and non-traffic-related events.

To use any kind of classifiers, first, features in data need to be specified. Becker et al. (2011) has proposed three set of features, temporal features, social features, topical features. Specifically, twitts are clustered based on their time. In this way, twitts in hour h are clustered together. Having clustered twitts, using feature mentioned above, they classified into event-related twitts and non event-related twitts. More precisely, Naive Bayes classifier and Support Vector Machine (SVD) are used as a classifier. Also, tf-idf is used as a way to assign weights to twitts. The authors claim that their method is scalable, and that does not require a prior knowledge of the number of cluster. However, to provide training data, they have done annotation task.

Moreover, Kumaran and Allan (2004) proposed a method using a text classifier method and entity based event detection. The authors use tf-idf method to assign weights to twitts. First, using an entity based event detection, stories i.e. events are detected and then using tf-idf, similarity between tweets is measured to increase cluster precision. However, the problem with the proposed method is that some events are categorized into multiple groups.

In another attempt, Sakaki et al. (2010) use SVD method to create a classifier to classify twitts into two groups of earthquake-related event and not earthquake-related event. For this, set of keywords are defined and then features for classifier are defined as number of words in a tweet, words in a tweet, the words before and after the query word. However, this model is based on this assumption that each twitt is associated with a time stamp.

In addition, another method is to use sentence classification. Zhang and Wallace (2015) proposed a method based on Convolutional Neural Networks (CNNs). According to this paper, sentences are shown as a matrix which its rows corresponds to tokens in the sentence and its columns is word vectors for each token. This representation makes us able to use CNN. Then, we can compute similarity between twitts and classify them. Still, this method is not applied to event detection problem. However, it is noteworthy that CNN is a costly computation and also word vectors of twitts are required to use this method. Regarding to this fact that, in twitter anylytics, we face a hugestreaming data it is neede that we come with some optimization for CNNs to use them in real time.

As an alternative to three sigma rule, I propose another method. This method is based on neural network and is similar to function approximation method used in Q-learning. Similar to

the original method, concept of windows is used. With help of windows with different length over twitts, frequency of entities is tracked. For instance, we have a window with length 5 minutes. The frequency of a word like "football" in the last 15 minutes in the last three windows is used to predict whether the twitts, related to that entity, are showing an event. In this way, frequency of entities is fed to neural network to determine whether an entity is bursting. However, like other supervised learning, this method needs some annotated data for training.

A Algorithm

Algorithm 1 Creating inverted indexes

for each $twitt \in twitts$ **do**
  if $twitt.entity \in InvertedIndexes$ **then**
    $Freq(InvertedIndexes.twitt.entity) \leftarrow Freq(InvertedIndexes.twitt.entity) + 1$
  else
    $Freq(InvertedIndexes.twitt.entity) \leftarrow 1$
  end if
  for each $window \in windows$ **do**
    if $windowfinished$ **then**
      Detect bursting entities
      Update the window parameters
      Restart the window time
      if $window$ is the largest $window$ **then**
        $InvertedIndexes \leftarrow Initialize$
      end if
    end if
  end for
end for

Algorithm 2 Detecting bursting entities

for each $entity \in \mathcal{I}nvertedIndexes$ **do**
  if $entity \in windows$ **then**
    for each $window \in windows$ **do**
      $\mu \leftarrow \frac{S_1}{S_0}$
      $\sigma \leftarrow \sqrt{\frac{S_2 - S_1^2}{S_0}}$
      $size \leftarrow InvertedIndexes[entity].size$
      if $size \geq 3 * \sigma + \mu$ AND $size \geq 10$ **then**
        $burstinentities \leftarrow entity$
      end if
    end for
  end if
end for

Algorithm 3 Updating windows parameters based on three sigma rule

Precondition: $window$, determines which widow gets updated.

for each $entity \in \mathcal{I}nvertedIndexes$ **do**
  if $entity \in window$ **then**
    $window.S_0 \leftarrow window.S_0 + 1$
    $window.S_1 \leftarrow window.S_1 + InvertedIndexes[entity].size$
    $window.S_2 \leftarrow window.S_2 + (InvertedIndexes[entity].size)^2$
  else
    $window.S_0 \leftarrow 0 + 1$
    $window.S_1 \leftarrow 0 + InvertedIndexes[entity].size$
    $window.S_2 \leftarrow 0 + (InvertedIndexes[entity].size)^2$
  end if
end for

Algorithm 4 Traffic event detection

Precondition: Set of traffic-related keywords should be defined.

for each $twitt \in twitts$ **do**
  if $twitt.entity \in InvertedIndexes$ **then**
    $InvertedIndexes.entity.freq \leftarrow InvertedIndexes.entity.freq + 1$
  else
    $InvertedIndexes.entity.freq \leftarrow 1$
  end if
  $OneHotVec \leftarrow CreateOneHotVec(twitt.text)$
  $InvertedIndexes.entity.addNewTwitt(OneHotVec)$
  for each $window \in windows$ **do**
    if window is over **then**
      $burstingEntities \leftarrow DetectBurstingEntities(window)$
      for each $burtingEntity \in burstingEntities$ **do**
        $prob \leftarrow Neural\_Net(burtingEntity)$
        if $prob \geq 0.7$ **then**
          $trafficEvents \leftarrow burtingEntity$
        end if
      end for
      Update the window parameters
      Restart the window time
      if *window* is the largest *window* **then**
        $InvertedIndexes \leftarrow Initialize$
      end if
    end if
  end for
end for

B Evaluation on 1 day data set

Number of Windows	Event Recall	Cluster Precision	F-Measure	Clusters	Matched Back	Detected Events
1	0.0.006	0.0.200	0.012	20	4	3
2	0.016	0.225	0.030	106	36	8
3	0.030	0.256	0.053	550	141	15
4	0.030	0.211	0.052	1503	317	15
5	0.030	0.207	0.052	4452	921	15
6	0.034	0.201	0.058	11169	2244	17
7	0.034	0.197	0.057	24317	4801	17
8	0.034	0.171	0.056	49968	8544	17
Baseline	0.038	0.014	0.020	8829	120	19

Table 2: Burst detection with sliding windows on 1 day dataset

References

Becker, H., Naaman, M., and Gravano, L. (2011). Beyond trending topics: Real-world event identification on twitter. *ICWSM*, 11(2011):438–441.

Kumaran, G. and Allan, J. (2004). Text classification and named entities for new event detection. In *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 297–304. ACM.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.

McMinn, A. J. and Jose, J. M. (2015). Real-time entity-based event detection for twitter. In *International Conference of the Cross-Language Evaluation Forum for European Languages*, pages 65–77. Springer.

Sakaki, T., Okazaki, M., and Matsuo, Y. (2010). Earthquake shakes twitter users: real-time event detection by social sensors. In *Proceedings of the 19th international conference on World wide web*, pages 851–860. ACM.

Zhang, Y. and Wallace, B. (2015). A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. *arXiv preprint arXiv:1510.03820*.