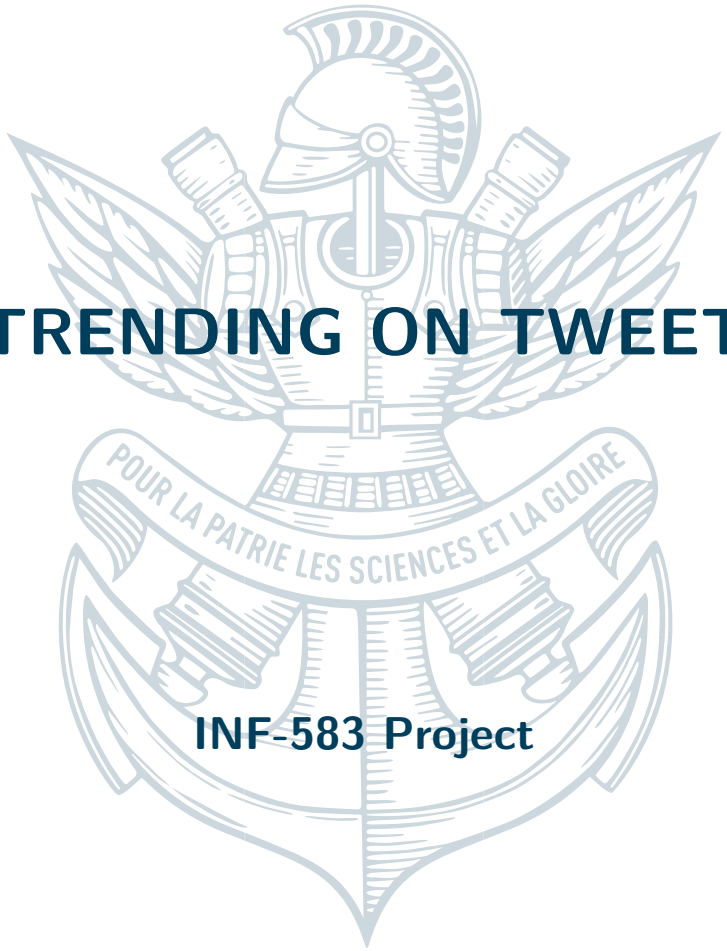



# TRENDING ON TWEET

## INF-583 Project

March 2021

Kunhao ZHENG, Yujia FU

# CONTENTS

# 1 INTRODUCTION

Nowadays, social media has became an important way for people to get news. Compared to the mainstream media, it has larger coverage of events and shows the pressing issues discussed by citizens, can bring awareness and break the bubble. So the event detection in social media is meaningful. However, there is a lot of spam, publicity and false news. What's more, people use unstructured, short text and informal language. So the task of event detection is not easy.

In this project, we use PySpark(Apache Spark in Python) to perform trending detection and analyse on tweet. The data contains tweets in English and French in the first week of February 2020. There are 9 steps in total. Due to the limit of the computational capacity, we use one week of French tweets for the first 4 steps and one day of English tweets for the rest steps.

All related codes could be found on this github repository DyeKuu/Trending-on-Twitter.

# 2 TRENDING DETECTION

## 2.1 Most frequent words

Listing 1: Python Spark setup

```
conf = SparkConf().setAppName("question1").setMaster("local[*]")
sc = SparkContext(conf=conf)
spark = SparkSession(sc)
lines = sc.textFile(','.join(FILES_PATH))
lines = lines.map(lambda line: json.loads(line))
```

In this part, we work on the French dataset. The code in Listing 1 shows how to set up Spark in Python, we load the data and convert them each to json format.

Listing 2: Filtering time range

```
date_time_begin_str = 'Sat Feb 1 20:00:00 +0000 2020'
date_time_end_str = 'Sun Feb 2 20:00:00 +0000 2020'
date_time_pattern = '%a %b %d %H:%M:%S %z %Y'
def filter_time_range(s):
    curr_datetime = datetime.datetime.strptime(s['created_at'], date_time_pattern)
    return curr_datetime > date_time_begin_obj and curr_datetime < date_time_end_obj

words = lines.filter(filter_time_range).map(lambda d: d['text']).flatMap(lambda s:
    s.strip().split())
```

To find out the most frequent word in a certain time range, we define two variables to limit the time range, `date_time_begin_str` and `date_time_end_str` shown in Listing 2. Then we define

a function `filter_time_range` to filter all the tweets sent in the time range. Afterwards we do a flatmap to convert the texts into words.

Listing 3: Most frequent word

```
most_frequent_word = words.map(lambda x: (x, 1)).reduceByKey(lambda a, b: a +
    b).max(key=lambda x: x[1])
```

The code in Listing 3 uses the map-reduce paradigm to count the occurrence of words. We first map each word to a tuple, which is itself along with unit count 1. Then we reduce all the tuple by key, which will take 2 objects with the same first element of tuple, and reduce them to 1 object using the rule define in `reduceByKey`. Finally, we extract the tuple with the biggest number of occurrence, thus the most frequent word. Note that before `max` the code realizes lazy-execution and will instantiate until the last step. We observe a quick execution during the intermediate steps.

The result is shown as follows.

`Most Frequent word from Sat Feb 1 20:00:00 +0000 2020 to Sun Feb 2 20:00:00 +0000 2020 is: RT with occurence 48409`

## 2.2 Removing stopwords

The result in the previous section is reasonable because when the tweet is in fact a retweet, it begins with the pattern "RT @[username]:". So the word "RT" will have a very high occurrence. We are going to remove the stop words then do again the word count.

We use the list of stop words in French provided on the Internet[1]. We found that the list is not complete in our specific case. So we extend the list by adding `['rt', ':', '?', ',', '.', '!','-', '_', '/', '«', '»', "c'est", "c'est", "j'ai", "j'ai"]`. Note that there are 2 encoding for the symbol single quote.

There are some fixed variables during the runtime, so it would be nice if we broadcast them. We broadcast notably 4 variables here: `date_time_begin_obj, date_time_end_obj, date_time_pattern, stop_words`

Then we redo the same word count to the filtered words, with the result shown as follows.

`Most Frequent word from Sat Feb 1 20:00:00 +0000 2020 to Sun Feb 2 20:00:00 +0000 2020 is: faire with occurence 2678`

## 2.3 Frequent seasonal words

We consider that some of the tweets are written in the evening, where some words frequently appear but not in the daytime. To find out theses words, our approach is to split the tweets into 2 categories: night and day, according to its time. We use 2 constants to define the night time range. `NIGHT_BEGIN = 18, NIGHT_END = 6`. We also broadcast these 2 variables. The function in Listing 4 shows how we retreive the night tweets.

[1] https://countwordsfree.com/stopwords

Listing 4: Filter night tweets

```
def isNight(s):
    curr_hour = datetime.datetime.strptime(s['created_at'],
        date_time_pattern_broadcast.value).astimezone(local_timezone_broadcast.value).hour
    return curr_hour >= NIGHT_BEGIN_BROADCAST.value or curr_hour <=
        NIGHT_END_BROADCAST.value
```

We work all French tweets in this task. We separately count each word's occurrence in 2 categories, then we sort by the difference between the word count at night and that in day. Finally we take top 10 words who have the biggest difference. The code in Listing 5 shows all the logics discussed above. The `leftOuterJoin` will join 2 word count RDDs, and extending the second element of the tuples, for which the function `calDiffAndReverse` will calculate the difference. Note that the function also reverse the position to facilitate the `sortByKey`.

Listing 5: Calculate word count difference

```
def calDiffAndReverse(x):
    if x[1][1] is None:
        return (x[1][0], x[0])
    return (x[1][0] - x[1][1], x[0])

result = night_words_count.leftOuterJoin(day_words_count).map(
    calDiffAndReverse).sortByKey(False).take(10)
```

The result is shown as follows. We can observe that, the words with biggest difference word count between night and day are reasonable. Like "vie", "demain", "soir", even "😂". Maybe people need to relax during the night.

`Top 10 most frequent words only in evening: [(885, 'vie'), (808, 'demain'), (802, 'faire'), (741, 'oui'), (722, 'veux'), (681, '😂'), (599, 'soir'), (572, 'twitter'), (570, 't'es'), (562, 'meuf')]`

## 2.4 Anomaly detection

### 2.4.1 • Point by point Poisson Model

To make anomaly detection in time serious, we use the Point-by-point Poisson Model introduced in the Technical report of tweet in 2015 [3]. We assume that the counts of tweets that contains a certain hashtag in a social data time series are Poisson-distributed around some average value, and then looking for unlikely counts according to the Poisson model shown below.

$$P(c_i; v) = \frac{v^{c_i} \cdot e^{-v}}{c_i!}$$

Where P is the probability of observing $c_i$ Tweets that contain a certain hashtag in the given time window, when the expected number of such Tweets is $v$. In our simple model, we use the count $c_{i-1}$ in the previous time window as the $v$. Then we check the unlikeliness of $c_i$ by looking at its distance with mean and the confidence interval of the count. We use a parameter $\eta$ to evaluate the unlikeliness.

$$c_i = \eta \cdot CI(\alpha, v) + v$$

Where $\alpha$ is the confidence level and CI() is the confidence interval. When $\eta$ exceeds a threshold $\eta_c$, we say that an anomaly is detected.

### 2.4.2 • Implementation on french tweet data set

Due to the limit of computational capacity, we use the smaller data set: tweets in French in the first week of February 2020. In order to get enough data, we want to choose a target hashtag with high total count. So first of all, using the code in Listing 6, we count the number of the hashtags mentioned in the data set and list the 20 most frequent hashtags. The resulted hashtags are shown in Fig 1.

Listing 6: Most Frequent hashtags

```
def get_all_hashtags(s):
    hashtags = s['entities']['hashtags']
    res = []
    for hashtag in hashtags:
        res.append((hashtag['text'], 1))
    return res

frequent_hashtags = lines.flatMap(get_all_hashtags).reduceByKey(lambda a, b: a +
    b).sortBy(lambda x: x[1], ascending = False).take(20)
```

```
Count of #Macron: 776          Count of #Mila: 691            Count of #Concours: 671        Count of #coronavirus: 610
Count of #LREM: 522            Count of #Paris: 469           Count of #GiletsJaunes: 464    Count of #iHeartAwards: 449
Count of #AsimForTheWin: 404   Count of #ASSEOM: 384          Count of #CONCOURS: 366        Count of #retraites: 352
Count of #Municipales2020: 347 Count of #Brexit: 330          Count of #FCGBOM: 303          Count of #AsliFans: 285
Count of #BB13OnVoot: 284      Count of #RT: 273              Count of #PSGMHSC: 270         Count of #SuperBowl: 267
```

Figure 1: Top 20 hashtags of tweets in French in the first week of February 2020

We choose '#coronavirus' as our target hashtag. We set time window = 2h and the confidence level alpha = 0.99. With the code in Listing 7, we firstly filter the tweets which contain the hashtag '#coronavirus'. Then we assign the time window to each tweet using the function 'get_time_period_for_row', note it as a property 'created_at_time_period'. We count the number of '#coronavirus' in each time interval, and calculate the $\eta$ with Poisson model introduced in 2.4.1 using the function 'get_eta'.

Listing 7: Anomaly detection of hashtag '#coronavirus'

```
tweet_with_hashtag = lines.filter(lambda s: filter_hashtag(s, target_hashtag)).map(lambda
    s: get_time_period_for_row(s, delta_time, date_time_begin_obj))
hashtag_trend = tweet_with_hashtag.map(lambda s: (s['created_at_time_period'],
    1)).reduceByKey(lambda a, b: a + b)
hashtag_trend_df = spark.createDataFrame(hashtag_trend,schema=['t',
    'hashtag_count']).toPandas()
hashtag_trend_df['eta1 (alpha = 0.99)'] = hashtag_trend_df.apply(lambda x: get_eta(x, alpha
    = 0.99), axis=1)
```

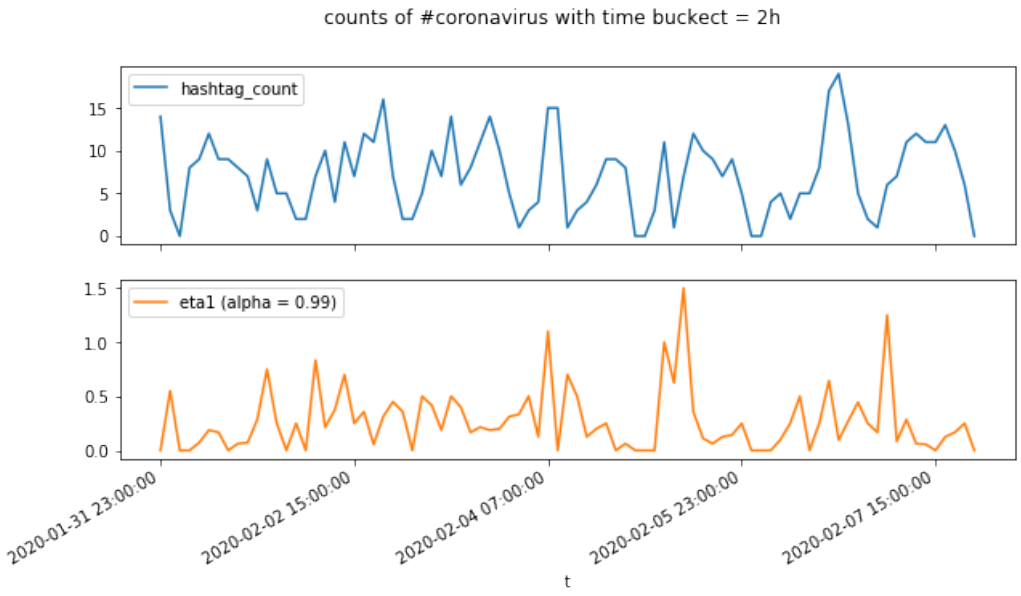



Figure 2: Mentions of "#coronavirus" per 2-hour intervals. For each point, $\eta$ is calculated based on the previous point, and plotted in yellow. In this case, $\alpha = 0.99$.

With the threshold $\eta_c = 1$, we detect three time points with anomaly: '2020-02-04 07:00:00', '2020-02-05 11:00:00', '2020-02-07 05:00:00'.

### 2.4.3 • Comparison of different parameters

The anomaly detection depends on the choice of two parameters values: $\alpha$ and the time interval for a single data point. We choose different values of these parameters and analyse their influences.

- Influence of $\alpha$

In order to analyse the influence of $\alpha$, we use the same data set and parameters as in 2.4.2, and change only the value of $\alpha$ into 0.6 and 0.3.

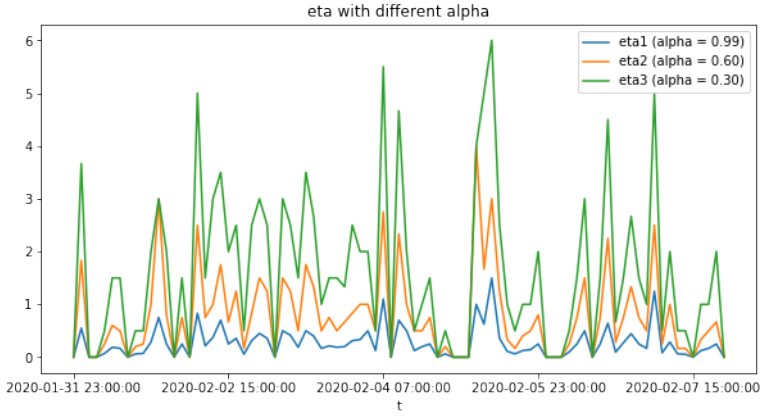



Figure 3: $\eta$ of "#coronavirus" per 2-hour intervals with different values of confidence level $\alpha$.

The $\eta$ of hashtag counts calculated with different $\alpha$ is shown in 3. With a lower confidence level $\alpha$, the $\eta$ increases. That's because the confidence interval becomes smaller and the real count is more propabable to be far away from the confidence interval. So more trends are detected, but the noise also increases.

- Influence of time interval

In order to analyse the influence of time interval, we use the same data set and parameters as in 2.4.2, and change only the value of time interval into 4h and 8h.

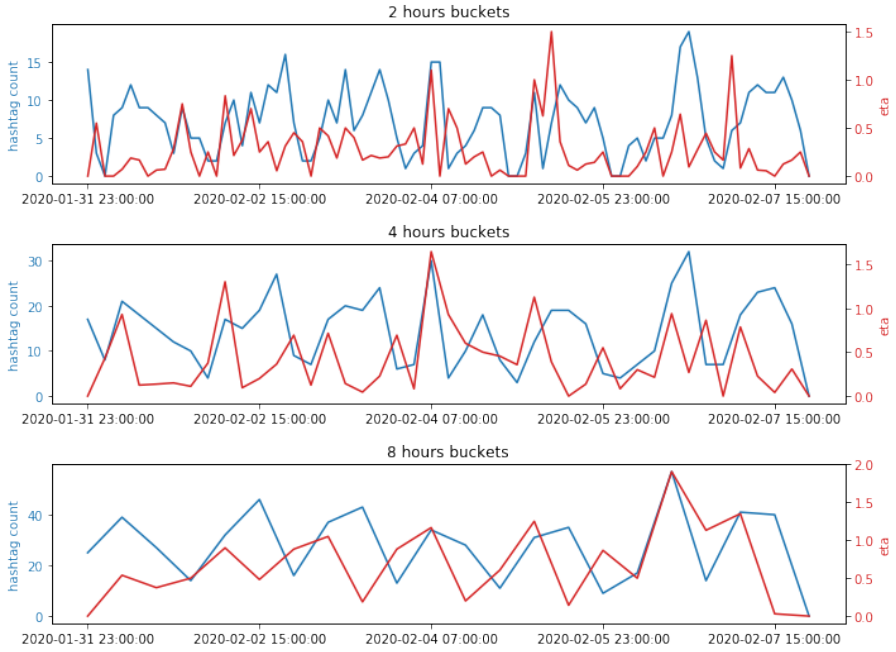



Figure 4: Mentions of "#coronavirus" in different size of time interval. For each point, $\eta$ is calculated based on the previous point, and plotted in red. In this case, $\alpha = 0.99$.

The $\eta$ of hashtag counts calculated with different time intervals is shown in 4. With a smaller bin size, more trends are detected, but with a worse precision. Because when we use smaller time interval, we will catch more detailed trend of counts, some are noises.

## 2.5 Frequent words describing an event

In order to use some helpful English NLP libraries, we change into the English tweets data set in the following 4 steps. Since the data is large, we use the tweets in English on 02/01/2020.

To extract the most frequent words describing an event, we detect event using the anomaly detection of one hashtag(as in 2.4), and then find out the most frequent words without stop words of the event related tweets in the event periods(as in 2.2).

Listing 8: Frequent Words describing an event

```
tweet_at_event = tweet_with_hashtag.filter(lambda s: s['created_at_time_period'] in
    event_intervals)
words_at_event = tweet_at_event.flatMap(lambda s: ps.stem(s['text']).strip().split())\
              .filter(lambda s: s not in stop_words)
most_frequent_words_at_event = words_at_event.map(lambda x: (x, 1))\
    .reduceByKey(lambda a, b: a + b)\
    .map(lambda s: (s[1], s[0]))\
    .sortByKey(False).take(20)
```

We choose the hashtag '#BiggBoss13' to detect related events. We set the time interval = 1h, $\alpha = 0.99$, $\eta_c = 1.1$. After the anomaly detection, we detect three event periods. In the code at Listing 8, 'tweet_with_hashtag' are tweets with hashtag '#BiggBoss13'. We filter them with the event periods to get tweets posted during the events, and we note them as 'tweet_at_event'. Then we extract the 20 most frequent words from 'tweet_at_event'. The resulted words that together describing the event '#BiggBoss13' are shown in Fig 5.

```
Count of #biggboss13: 380      Count of #bb13: 175           Count of #asimriaz: 93        Count of support: 63
Count of #asimforthewin: 60    Count of asim: 49            Count of fans: 42             Count of journey: 42
Count of #shehnaazgill: 41     Count of love: 40            Count of winner: 40           Count of #biggboss: 36
Count of all.: 34              Count of winner!!!: 34       Count of doubt: 34            Count of sureshot: 33
Count of proved: 33            Count of #siddharthshukla: 33 Count of praise: 33          Count of boss: 32
```

Figure 5: Top 20 frequent words to describe the event '#BiggBoss13'.

## 2.6 Event time frame

Now we want to get the time frame of the event detected in 2.5. With the code in Listing 9, we firstly extract all the sentences for 'tweet_at_event' grouped by event period, and note them as 'sentences_at_event'. Then we make a summarize for sentences at each period with the function 'summarize'. In the summarize process, we clean the tweet text by the function 'filter_tweet_special_text', which remove the words 'RT', the username indicator that starts with " and the url start with 'http'. To do the summarize, we use the WordFrequency Algorithm: we count the word frequency in the text and calculate the score of each sentence by summing all its words' frequency, then take the two sentences with the highest scores to make a summarize. The summarize would be the introduction for one period in the event time frame.

Listing 9: Time frame of an event

```
sentences_at_event = tweet_at_event.map(lambda s: (s['created_at_time_period'],
    s['text'])).reduceByKey(lambda a,b: a + '. ' + b)
sentences_at_event_list = sentences_at_event.collect()

summary_at_event = {}
for item in sentences_at_event_list:
    event_time = item[0]
    text = item[1]
    summary = summarize(text, n_top_sentence, tweet_stop_words = stop_words)
    summary_at_event[event_time] = summary
    print('At {}: \n {}'.format(event_time, summary))
```

```
    print()

def filter_tweet_special_text(word):
    if word == 'RT'or word.startswith('@') or word.startswith('http'):
        return False
    return True
```

The time frame for the '#BiggBoss13' event is:

- At 2020-02-01 07:00:00: All praise to the fans of Asim who have proved in his journey in #BiggBoss13 that their love & support has playe... Vikas Gupta entered #BiggBoss13 in support of Sid & tried to play dirty game by dragging innocent girl Sharuti T.....
- At 2020-02-01 16:00:00: #BiggBoss13: Prior to #SalmanKhan 's #WeekendKaVaar, #Asim-Riaz 's fans show support & trend #AsimForTheWin' -. Vikas Gupta entered #BiggBoss13 in support of Sid & tried to play dirty game by dragging innocent girl Sharuti T...
- At 2020-02-01 17:00:00: #BiggBoss13 #BiggBoss #Asim #AsimRiaz #Shukla #SiddharthShukla #ShehnazKaur..... Sana was a potential winner of #BiggBoss13 but because of obsession with Siddhart Shukla she looks really bad, flipper & no...

## 2.7 Location of event

To get the location of event, we use the 'location' property of the tweet. With the code in Listing 10, we count the number of locations for all the 'tweet_at_event'. The most frequent locations of event '#BiggBoss13' are shown in Fig 6. We can deduce that the event happened in India.

Listing 10: Location of an event

```
locations_at_event = tweet_at_event.filter(lambda s: 'user' in s and 'location' in
    s['user'])\
    .map(lambda s: (s['user']['location'], 1))\
    .reduceByKey(lambda a,b: a + b)\
    .sortBy(lambda x: x[1], ascending = False).take(20)
```

```
Count of None: 246              Count of India: 35              Count of Mumbai, India: 10      Count of New Delhi, India: 9
Count of Hyderabad, India: 5    Count of india: 4               Count of Nepal: 3               Count of Gondal, India: 2
Count of Gujarat, India: 2      Count of मुंबई, भारत: 2         Count of Bengaluru, India: 2    Count of Dhaka, Bangladesh: 2
Count of Delhi, India: 2        Count of Delhi: 2               Count of Tikathali, Lalitpur: 1Count of United Kingdom: 1
Count of Gurgaon, India: 1      Count of Dehradun : 1           Count of Punjab, India: 1       Count of Tata Steel City, Jamshedpur: 1
```

Figure 6: Top 20 frequent locations of the event '#BiggBoss13'.

## 2.8 Sentiment around the event

We use Vader(Valence Aware Dictionary for sEntiment Reasoning) [2] to analyse the sentiment of text. Vader is a library attuned to sentiments expressed in social media. It calculates a score between -1 to 1 for a sentence. A higher score means positive sentiment and a lower one means negative.

With the function 'get_sentiment_scores' in Listing 11, we do sentiment analysis for every sentence in the tweets of the event '#BiggBoss13'. We firstly split the text into sentences, and then remove some special tweet characters with the function 'filter_tweet_special_text'. Finally we do sentiment analyse with the Vader libarary. The histogram of sentiment scores for all the sentences are shown in Fig 7. Most of the tweets are neutral and there are more positive tweets than negative tweets.

Listing 11: Sentiment analysis of an event

```
def get_sentiment_scores(text):
    sentiment_scores = []
    sentence_list = nltk.sent_tokenize(text)
    sentence_list = [newsent + '.' for sent in sentence_list for newsent in
        sent.split('\n') ]
    sentence_list = [' '.join([word for word in sent.split(' ') if
        filter_tweet_special_text(word)]) for sent in sentence_list]
    analyzer = SentimentIntensityAnalyzer()
    for sentence in sentence_list:
        vs = analyzer.polarity_scores(sentence)
        sentiment_scores.append(vs['compound'])
    return sentiment_scores
```

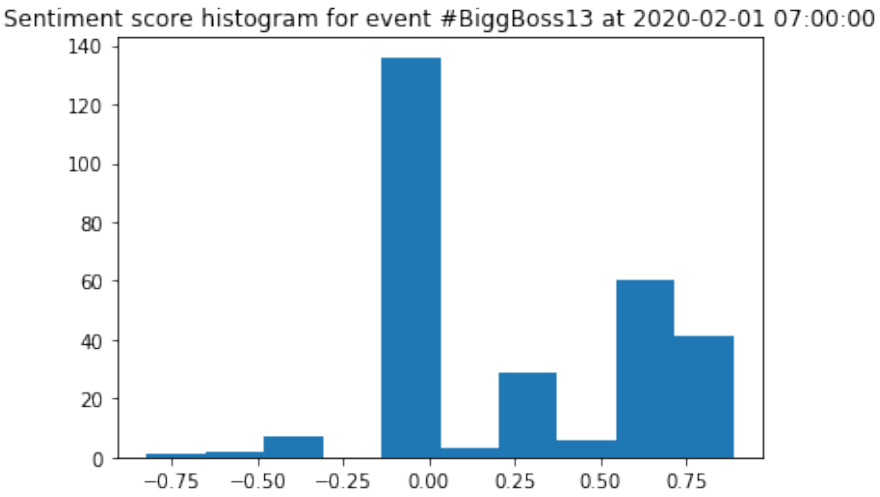



Figure 7: Histogram of sentiment scores of tweets in the event '#BiggBoss13'. The score is between -1 to 1. A higher score means positive sentiment and a lower one means negative.

## 2.9 Cluster tweets

We work on a subset of the original English dataset. We take only the tweets in 2020-02-01. Then we first take a look at the top-20 hashtags. Then we extract the tweets only involving the top-5 hashtags, which correspond to top-5 events of that day. We will try to cluster these tweets.

To do tweets clustering, we must convert the tweets in text format to vectors. Due to the tweet's characteristics that it has a small length. Our first attempt is to use sentence embedding in FLAIR

framework[1] and plug it into the spark RDD map-reduce. However, we encountered mysterious error whether we broadcast the object for FLAIR framework or not. So we abandon this plan.

Our second attempt is to train a Word2Vec embedding from scratch based on our twitter dataset. This is more feasible because Spark has a bulit-in Word2Vec model framework.

Listing 12: data preprocessing

```
def clean_tweet(t: str):
    t = t.replace('\n','').strip()
    t = re.sub("(@[A-Za-z0-9]+)|([^0-9A-Za-z \t])|(\w+:\/\/\S+)"," ",t).split()
    if t[0] == 'RT':
        t = t[1:]
    return t
tweet_with_hashtag_cleaned = tweet_with_hashtag.map(lambda s: (clean_tweet(s[0]), s[1]))
```

The code in Listing 12 shows how we do the data preprocesssing to feed in the Word2Vec model. Basically, we want to remove all the pattern like url, @[username], 'RT' tag and hashtags. For example, if the tweet before is `'RT @Tha5SOSVote: 50 REPLIES = 50 VOTES \n\n#iHeartAwards #BestCoverSong #DancingWithAStranger @5SOS https://t.co/C6LB3JhqmN'`, then we will expect it to become `['50', 'REPLIES', '50', 'VOTES', 'iHeartAwards', 'BestCoverSong', 'Dancing WithAStranger']` after the preprocessing. This is exactly done by the function `clean_tweet`.

Listing 13: Training Word2Vec

```
from pyspark.ml.feature import Word2Vec
word2Vec = Word2Vec(vectorSize=2, minCount=0, inputCol="text", outputCol="result")
model = word2Vec.fit(tweet_with_hashtag_cleaned_df)
result = model.transform(tweet_with_hashtag_cleaned_df)
```

The code above trains a Word2Vec model with feature space dimension = 2. The reason is that we cannot afford a large computation cost and it is easier for visualization in dimesion = 2. To obtain the embedding of a sentence, the method `model.transform` automatically takes the average of word vector in a sentence and returns it as the embedding of the sentence. The model adds a column `result` to store the embedding in the original dataframe.

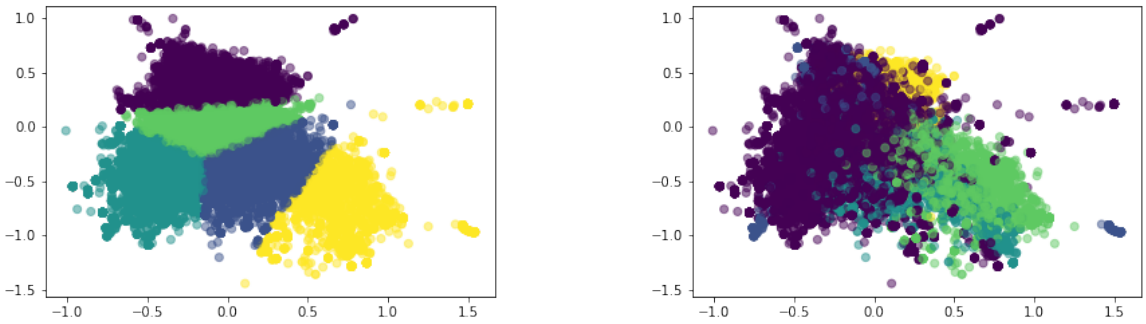
Listing 14: K-means Clustering

```
from pyspark.ml.clustering import KMeans
kmeans = KMeans().setK(5).setSeed(1234).setFeaturesCol('result')
model = kmeans.fit(result.select('label', 'result'))
centers = model.clusterCenters()
transformed = model.transform(result.select('label', 'result'))
```

The code in Listing 14 shows that we use k-means to do clustering for the sentence vector. It will add a column `prediction` to store the predicted label.

Although k-means clustering can capture the cluster on the lower-right, it is not so satisfying. We think of several possible reasons:

- The word vector space dimension is 2, thus too small. We cannot well separate sentences with different meaning.
- We observe that the tweet for the top event has roughly 10 times more than the second top



(a) K-means clustering

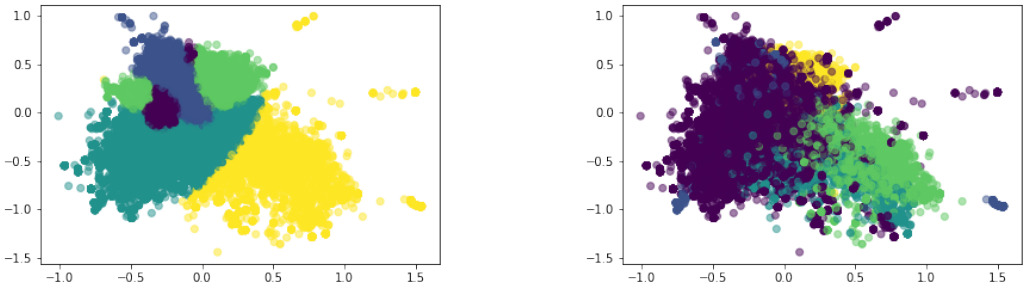
(b) Ground Truth

Figure 8: Comparison of clustering on top-5-event tweets

event. Our dataset is not very balanced. Some hashtags are relavant.

- K-means is a linear model so it cannot deal with very well unlinear boundary.

To mitigate the limit of linear model, we tried Gaussian Mixture Model to do the clustering. The Figure 9 shows that GMM is more suitable for clustering and it can capture both the boundary of the clusters on the upper-right and lower-right.

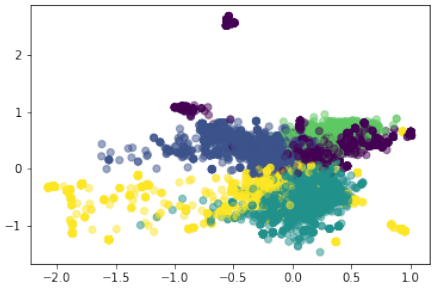


(a) GMM clustering

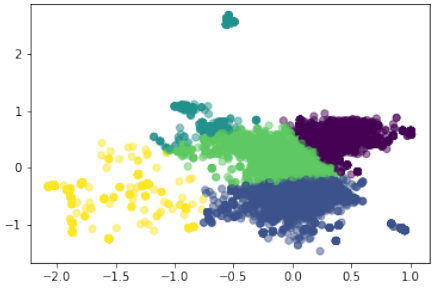
(b) Ground Truth

Figure 9: Comparison of clustering on top-5-event tweets

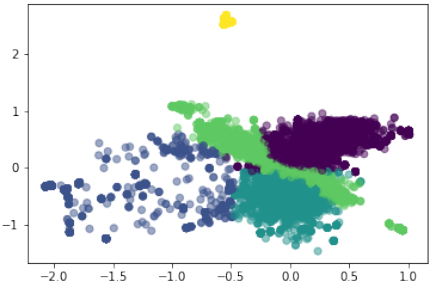
We also observe that the hashtags that we have chosen may be highly relevant, like #AsimForTheWin and #AsimRiaz. We will choose another 5 topics semantically well separated and of the similar amount. We re-conduct all our experiments and the results are shown in Figure 10. It turns out that in dimension = 2, although both can capture some cluster structure, they are not very satisfying but still, GMM outperforms K-means.




(a) Ground Truth




(b) K-means clustering




(c) GMM clustering

Figure 10: Comparison of clustering

# REFERENCES

[1] Alan Akbik et al. “FLAIR: An easy-to-use framework for state-of-the-art NLP”. In: *NAACL 2019, 2019 Annual Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*. 2019, pp. 54–59.

[2] CHE Gilbert and Erric Hutto. “Vader: A parsimonious rule-based model for sentiment analysis of social media text”. In: *Eighth International Conference on Weblogs and Social Media (ICWSM-14). Available at (20/04/16) http://comp. social. gatech. edu/papers/icwsm14. vader. hutto. pdf.* Vol. 81. 2014, p. 82.

[3] S. Hendrickson et al. “Trend detection in social data”. In: *Technical report, Twitter Inc.* June 2015.