Topic 5:
Implement Twitter Search Engine & Data Processing

1. Preface

The report will describe how we implement a twitter search engine and collect the tweets of SP500 companies CEOs , Then apply filter to fetch tweets related to companies' earnings or the whole economic activities . So to predict the trend of SP500 index.

2. Twitter Search Engine

We first use the twitter API as our code base to fetch the data from CEOs' twitter account , However , we then find that the twitter API has too many limitations , for example:

a. If api.user_timeline method is used to collect tweets from a particular twitter account , only the most recent 3240 tweets of a specific user can be collected.
b. If api.search method is used to search the key words from all accounts , only the past 10 days data will be displayed.
c. If streaming api is used , only the up-to-date data can be displayed , the historical data can not be fetched.

As a result , we develop our own twitter search engine.

The input is quite simple , we just need to specify the twitter accounts we want to collect the tweets data from and also specify a timeframe (for example , from 2015,01,01 to 2017,01,01), then the engine will collect all tweets from all users within this timeframe.

Step 1: Run the multi_scrap_queue.py

This python script will fetch the Twitter_Top_50.csv as inputs , user can specify the timeframe and the NUMBER_OF_WORKERS in the file for usage. (Notice that the NUMBER_OF_WORKERS should not exceed the number of logical processors in your computer). The script will only get all the ids of tweets data , and place them under ids/ folder , we will use the next script to fetch metadata of these ids.

Code explain:

```
if __name__ == '__main__':
    jobs = multiprocessing.JoinableQueue()
    if len(users) < NUMBER_OF_WORKERS:
        NUMBER_OF_WORKERS = len(users)
    for w in range(NUMBER_OF_WORKERS):
        p = multiprocessing.Process(target=run, args=(jobs,start,end))
        p.daemon = True
        p.start()
    for user in users:
        jobs.put(user)
    jobs.join()
    print('all jobs done here')

    # Let the worker die
    for w in range(NUMBER_OF_WORKERS):
        jobs.put(None)
```

To expedite the crawling process , we implement multiprocessing and queue , the users are the twitter accounts we want to handle , we will put these users in the queue for the workers to fetch . The NUMBER_OF_WORKERS is the subprocesses number to do the crawling job . Everytime a worker will fetch a user in the queue and do the run method to collect data in the (start,end) timeframe.

Notice that we use poison to kill all our subprocesses after all our jobs are done.

In every run method , there is a while loop to get the user from the queue , then the selenium.webdriver is used to control the web browser. The date timeframe is divided into 1 day partition (2015/01/01 - 2015/01/02) Then the user twitter accounts name and date will then form a url , for example:
https://twitter.com/search?f=tweets&vertical=default&q=from%3Afacebook%20since%3A2015-01-01%20until%3A2015-01-02include%3Aretweets&src=typd

This url will query for user(facebook) since 2015-01-01 to 2015-01-02 .

Then we use css selector to fetch the values of “href” attribute , which is the ids of tweet data.

```
for tweet in found_tweets:
    try:
        id = tweet.find_element_by_css_selector(id_selector).get_attribute('href').split('/')[-1]
        ids.append(id)
    except StaleElementReferenceException as e:
        print('lost element reference', tweet)
```

We will then increment the date by one , and keep doing the same process until all tweets ids of specific timeframe has been crawled.

Notice that each time when we done crawling data for a specific user , we need to call queue.task_done() to drop the done job in the queue.

Step 2: Run the get_metadata.py

This python script will use ids in ids/ folder as input , then use twitter api's api.statuses_lookup to fetch all metadata of tweets data , and then output a csv file which contains user(twitter account screen name) , created_at (the tweets creation date),text(the raw text) as columns.

We use api.statuses_lookup at this time because that now we have all the ids of tweets data , and it is more efficient to use this method to get all metadata of our tweets rather than crawling them in the web pages. Notice that with the ids/ , we can get historical tweets data from twitter , we are still limited by the twitter API , However , the limitation is not that stringent when fetching with tweets id.

GET statuses/lookup

Returns fully-hydrated Tweet objects for up to 100 Tweets per request, as specified by comma-separated values passed to the `id` parameter.

This method is especially useful to get the details (hydrate) a collection of Tweet IDs.

GET statuses / show / :id is used to retrieve a single Tweet object.

Resource Information

Response formats	JSON
Requires authentication?	Yes
Rate limited?	Yes
Requests / 15-min window (user auth)	900
Requests / 15-min window (app auth)	300

Above is an intercept from twitter API doc. We can see that with (user auth) , we can get 100 tweets 900 times in 15 min , that is to say , we can fetch 90000 tweets in 15 min which is quite fast in general use case.

Code explain:

```
auth = tweepy.OAuthHandler(keys['consumer_key'], keys['consumer_secret'])
auth.set_access_token(keys['access_token'], keys['access_token_secret'])
api = tweepy.API(auth)
```

We need this section to authenticate with twitter API , one can apply online in https://developer.twitter.com/ .

```
for go in range(i):
    print('currently getting {} - {}'.format(start, end))
    sleep(1)  # needed to prevent hitting API rate limit
    id_batch = ids[start:end]
    start += 100
    end += 100
    tweets = api.statuses_lookup(id_batch)
    for tweet in tweets:
        entry = tweet._json
        t = {
            "user": entry["user"]["screen_name"],
            "created_at": entry["created_at"],
            "text": entry["text"],
        }
        results.append(t)
```

In this section, we use api.statuses_lookup to fetch user(twitter account screen name) , created_at (the tweets creation date), text(the raw text) attributes of all tweets data, notice that we need to sleep(1) for each iteration to prevent hitting the API rate limit.

3. Tweets Data Filtering

In Order to get topic related tweets data , we use regular expression to filter our text data, we will briefly introduce regular expressions then show how to apply it in our code.

Regular expressions (called REs, or regexes, or regex patterns) are essentially a tiny, highly specialized programming language embedded inside Python .Using this little language, you specify the rules for the set of possible strings that you want to match.

The first metacharacters we'll look at are [and]. They're used for specifying a character class, which is a set of characters that you wish to match. For example, [abc] will match any of the characters a, b, or c .

You can match the characters not listed within the class by complementing the set.For example, [^5] will match any character except '5'.

the most important metacharacter is the backslash, \. It's used to escape all the metacharacters so you can still match them in patterns; for example, if you need to match a [or \, you can precede them with a backslash to remove their special meaning: \[or \\.

If the regex pattern is a string, \w will match all the characters marked as letters in the Unicode database.

\d Matches any decimal digit; this is equivalent to the class [0-9].

The final metacharacter in this section is . . It matches anything except a newline character.

| Alternation, or the "or" operator. If A and B are regular expressions, A|B will match any string that matches either A or B.

So much for the basic concepts for regular expression , we will now talk about some assertions that will use in our code.

Lookahead and lookbehind, collectively called "lookaround", are zero-length assertions just like the start and end of line, and start and end of word anchors . Lookaround allows you to create regular expressions that are impossible to create without them, or that would get very longwinded without them.

Negative lookahead is indispensable if you want to match something not followed by something else. For example , q(?!u) matches a q that is not followed by a u.

Similarly Positive lookahead means you want to match something followed by something else. For example , q(?=u) matches a q that is followed by a u.

Lookbehind has the same effect, but works backwards. For example , (?<!a)b matches a "b" that is not preceded by an "a".

In our code , both lookahead and lookbehind assertion will be used.

For example , If we want to get all tweets text data having the keyword 'company' , we can use the following regular expression:

(?<![\w\d])company(?![\w\d])

Which means that we want to match a string with 'company' , but we don't want any alpha characters (\w) or decimal numbers (\d) precede or after it. So that we can get the text data with exactly "company" word inside it.

Step 3: Run the preprocessing.py

This python script will use regular expression to filter all tweets text data , and only get the topic related data for further processing.

Code explain:

```
patterns = '(?<![\w\d])\$SPX(?![\w\d])' \
           '|(?<![\w\d])\#SPX(?![\w\d])' \
           '|(?<![\w\d])S&P 500(?![\w\d])' \
           '|(?<![\w\d])SP500(?![\w\d])' \
           '|(?<![\w\d])SANDP 500(?![\w\d])' \
           '|(?<![\w\d])S&amp;P 500(?![\w\d])' \
           '|(?<![\w\d])Company(?![\w\d])'  \
           '|(?<![\w\d])company(?![\w\d])' \
           '|(?<![\w\d])stock(?![\w\d])' \
           '|(?<![\w\d])stocks(?![\w\d])' \
           '|(?<![\w\d])Stock(?![\w\d])' \
           '|(?<![\w\d])Economic(?![\w\d])' \
           '|(?<![\w\d])economic(?![\w\d])' \
           '|(?<![\w\d])companies(?![\w\d])' \
           '|(?<![\w\d])Companies(?![\w\d])' \
           '|(?<![\w\d])market(?![\w\d])' \
           '|(?<![\w\d])markets(?![\w\d])'

filter = tweets_df['text'].str.contains(patterns,regex=True)
tweets_df = tweets_df[filter]
```

The code above will match tweets text contains any keywords in the patterns variable. Then output only keywords related tweets text tuples.

4. Cleaning data

Simplifying structure of sentence

The tm package provides a function tm_map() to apply cleaning functions to an entire corpus, making the cleaning steps easier.

tm_map() takes two arguments, a corpus and a cleaning function. Here, removeNumbers() is from the tm package.

Steps:

1.Converting Text into Lowercase.

2.Removing Punctuation Marks.

3.Removing any Numerical value from the tweets.

4.Removing stop words.

5.Removing Unwanted URLs.

6.Removing Whitespaces.

7.Clearing repetition of same words.

5. Visualization of dataset

Wordcloud

1.Word clouds add simplicity and clarity. The most used keywords stand out better in a word cloud

2.Word clouds are a potent communication tool. They are easy to understand, to be shared and are impactful

3.Word clouds are visually engaging than a table data

steps to create word clouds in R

Step1: Install and load the required packages

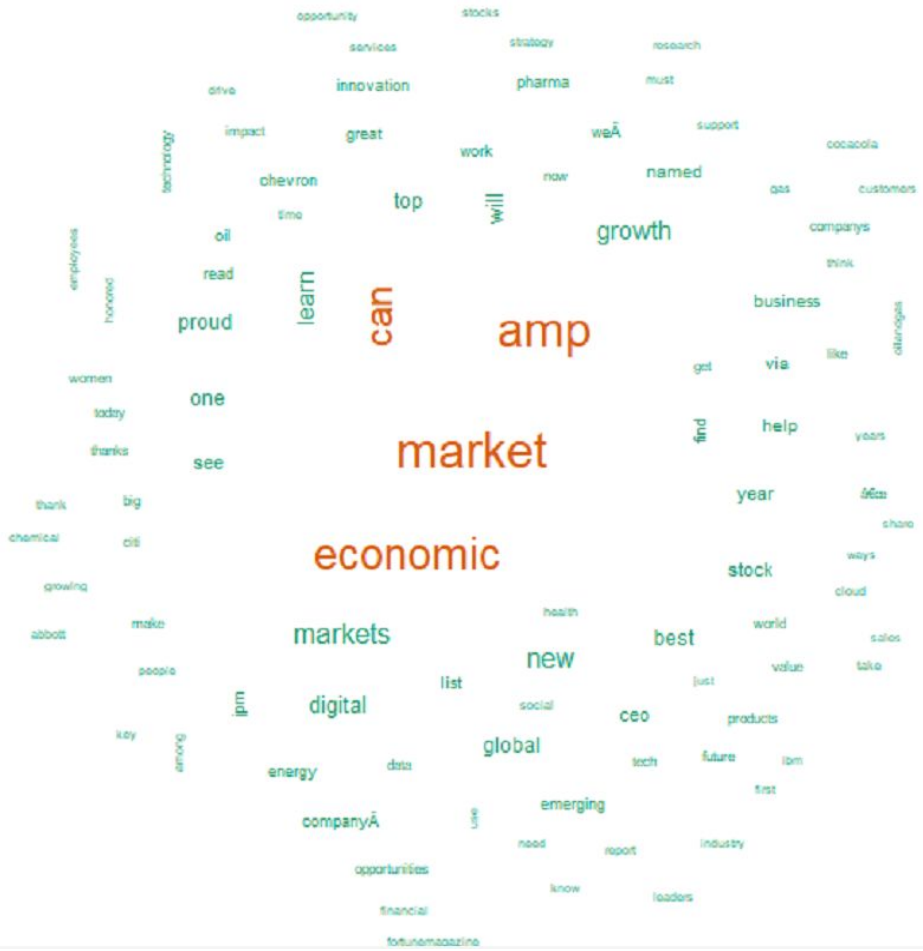
```
library(tm)
library("wordcloud")
library("SnowballC")
```

Step 2: Text mining
Read entire text into workspace.then it is time to clean data.Transformation is performed using tm_map() function to replace.

Step 3: Build a term-document matrix
Document matrix is a table containing the frequency of the words. Column names are words and row names are documents.

Step4: Generate the Word cloud

Picking out words whose occurrence is higher than 75



The above word cloud clearly shows that “market” “economic” are most important words.

6. Sentimental Analysis

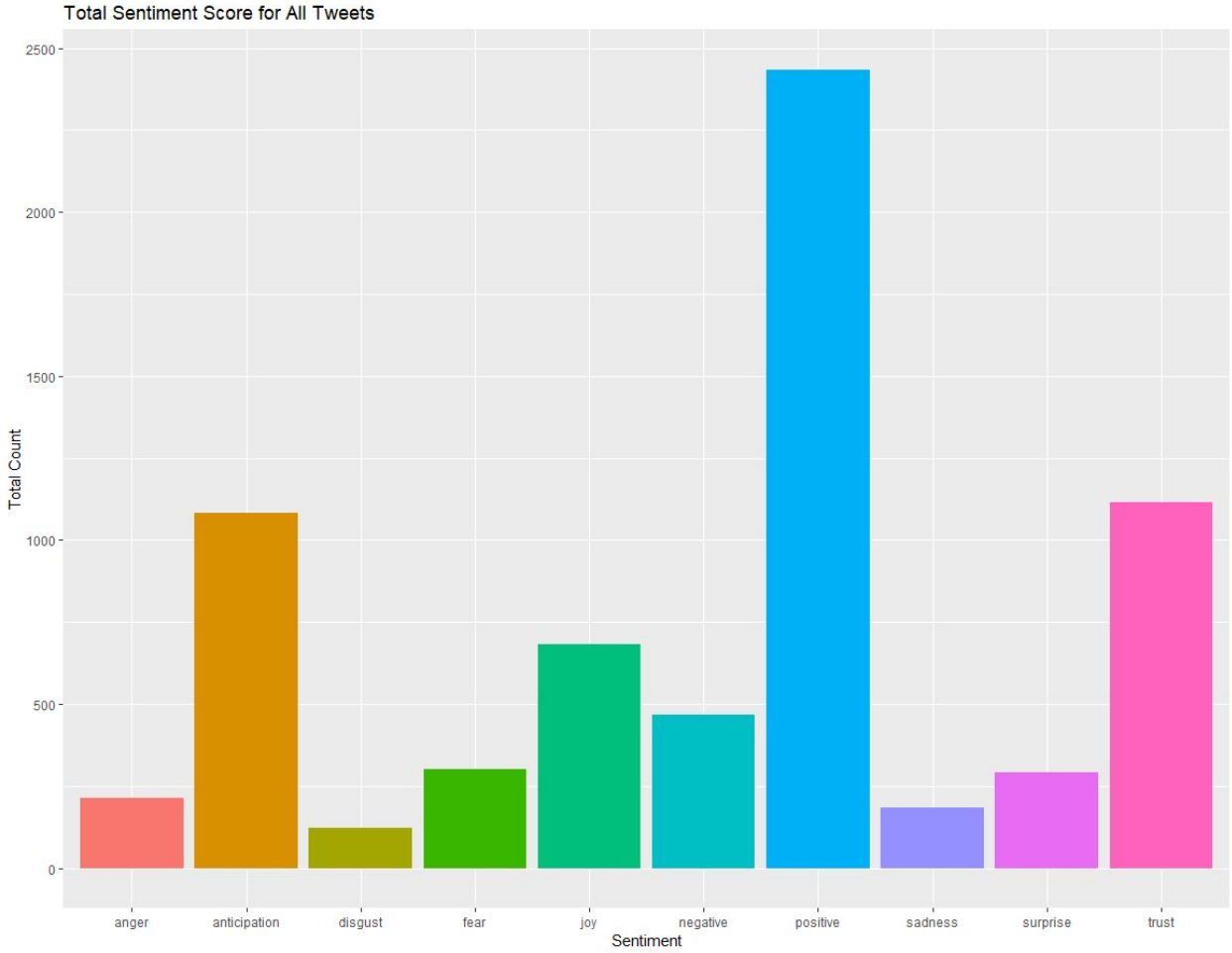
6.1 Counting Sentiment Score

The get_nrc_sentiment implements Saif Mohammad’s NRC Emotion lexicon. According to Mohammad, “the NRC emotion lexicon is a list of words and their associations with eight emotions (anger, fear, anticipation, trust, surprise, sadness, joy, and disgust) and two sentiments (negative and positive)” (See http://www.purl.org/net/NRCemotionlexicon). get_nrc_sentiment function returns a data frame in which each row represents a sentence from the original file. The columns include one for each emotion type was well as the positive or negative sentiment valence.

```
Sentiment <- get_nrc_sentiment(all_tweets_related$clean_text)
alltweets_senti <- cbind(all_tweets_related, Sentiment)
```

After performing above code,we have 10 types emotion count for each tweet.

Twitter volume in each sentiment category



Through above graph, the numbers of positive attitude much more than other attitude,trust and anticipation and attitude account for a large part in sentimen.

6.2 Making graph for Sentiment Over Time

S&P500 from 2015 to 2017


```
all_tweets_related$created_at <- mdy_hm(all_tweets_related$created_at)
all_tweets_related$created_at <- with_tz(all_tweets_related$created_at, "America/New_York")
```

In order to use cut() function to divide time into specific intervals, timestamps of the tweets must be converted to the same time zone. Function with_tz() returns POSIXct date-time objects.

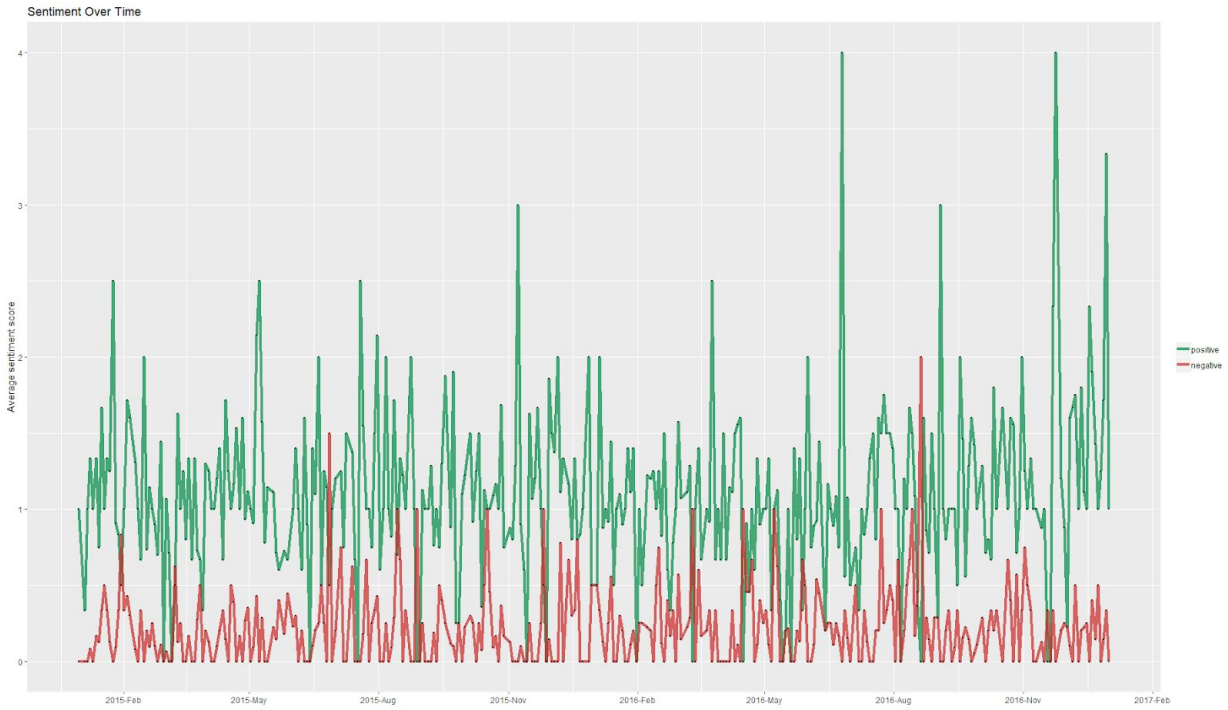
```
posnegtime <- alltweets_senti %>%
  group_by(created_at = cut(created_at, breaks="2 days")) %>%
  summarise(negative = mean(negative),
            positive = mean(positive)) %>% melt
```

Using created_at as variables

```
names(posnegtime) <- c("created_at", "sentiment", "meanvalue")
posnegtime$sentiment = factor(posnegtime$sentiment,levels(posnegtime$sentiment)[c(2,1)])
```

Creating posnegtime table in which row contains created time,sentiment value and mean value. Drawing point of mean value every two days. There are two lines,one is positive the other is negative. They both have point of mean value at same date.

Sentiment Over Time



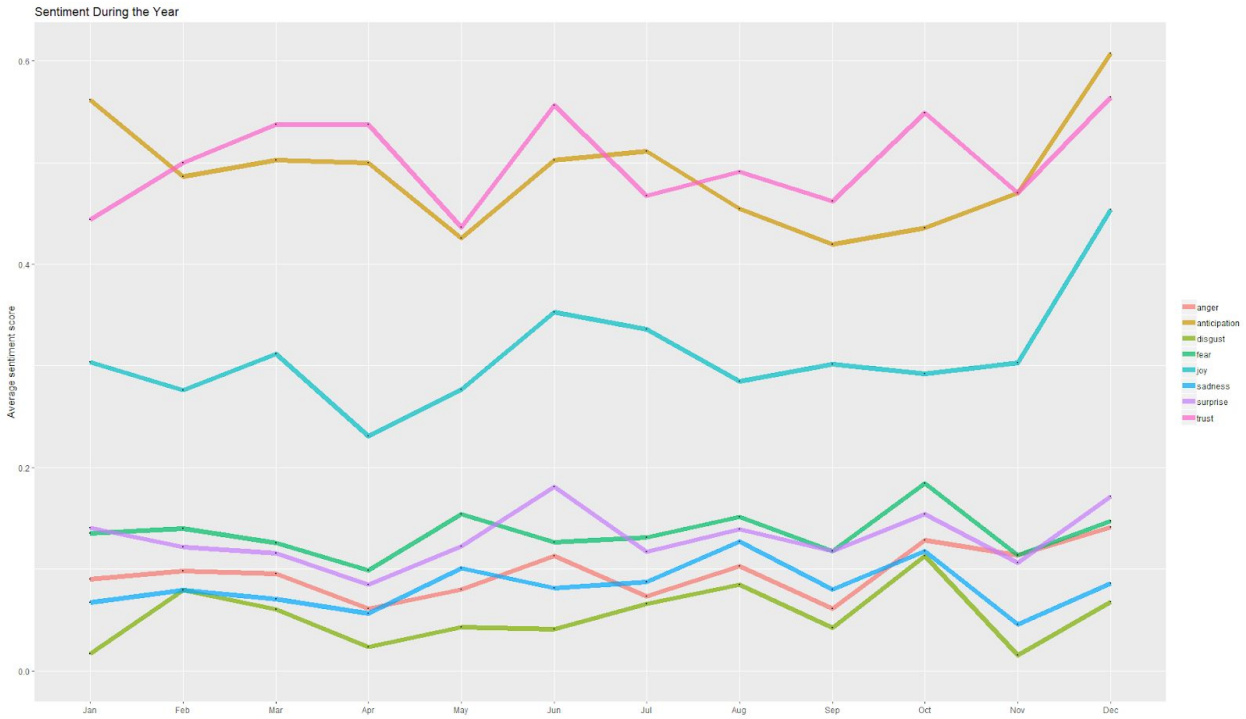
X axis is time from 2015-Feb to 2017-Feb. The tendency of positive is stronger than negative. At 2015-july the tendency of positive became more active,revealing the attitude of people towards S&P500 is more positive.

6.3 Making graph for Sentiment During the Year

```
alltweets_senti$month <- month(alltweets_senti$created_at, label = TRUE)
monthlysentiment <- alltweets_senti %>% group_by(month) %>%
```

Using moth as variables.grouping all data row by same month,calculating sentiment count for each tweet by mean function.
We select eight types of emotion to analyze over time. They are anger,anticipation,disgust,fear, joy,sadness,surprise,trust.

Sentiment During the Year



Apparently, from August the attitude of joy and anticipation show upward trend.if we compare this graph with S&P 500 graph, it clearly shows that with S&P 500 tend to upward people start to express joy and anticipation, which also tend to upward.

7. Conclusion

According to the comparison between S&P 500 and graphs for Sentiment During the Year and Sentiment Over Time. we could make a conclusion that in 8 emotion types, joy and anticipation could reveal the rising tendency of S&P 500. By capturing tweets which are sent by twitter accounts of CEO in top 500 companies, we could get a lot of information about S&P 500. Through filtering key words which we set up, more accurate data which we need will be produced.Based on this accurate data, we could get a relationship between sentiment and tendency of S&P 500 by analyzing sentiment during the year. On August attitude of joy and anticipation tend to more active and keep rising, at same time S&P 500 also tend to upward, so we could conclude that sentiment in tweets is related to tendency of S&P 500 reflecting the S&P 500 tendency in some extent.