Stock Price and Social Media Correlator

Asif A, Amin V, Hetal C, Johnny Y

---

1. Idea

Stock prices are a reflection of general market demand and expectation of a company's performance. Analysts have tried a variety of angles, both causal and correlative, to get an edge on the market and predict prices. In this project, we want to explore the relationship between Twitter mentions of a stock and the stock price. In particular, we will look at the 30 stocks that make up the Dow Jones Industrial Average (DJIA), one of the most widely referenced indexes in the US and globally. Our aim will be to test how well Twitter mentions can help predict the movements of the Dow Jones Industrial Average.

The problem is particularly ripe for exploring technologies in storage and retrieval for a few reasons. First, twitter chatter offers a constant stream of discussion around a company. To avoid noise from non-company related mentions, we will track ticker symbol mentions that will help to focus our data set. A highly-traded company (like Apple) receives 5-10K mentions per day based on our current research and estimates. Stretching this across the entire DJIA gives a high end of daily mentions of ~300K. Two, stock data is a more constant stream of minute by minute price information for 6.5 hours, which translates into ~12K data points per day across the DJIA. By bringing these two streams together, we plan to train an machine learning model to test how well we can predict the DJIA from twitter data.

2. Data Sources

2.1. Twitter Data

We are looking to mine twitter data to find the number of mentions for various companies and do some level of sentiment analysis.

The twitter APIs are quite robust and flexible in allowing for searching for tweets based on relevant filters.
For example, most tweets related to the stock ticker for Yahoo! are tagged with “$YHOO”. Twitter has 2 APIs we can use:

1. Streaming API: https://stream.twitter.com/1.1/statuses/filter.json?track=YHOO
2. Search API: https://api.twitter.com/1.1/search/tweets.json?q=YHOO

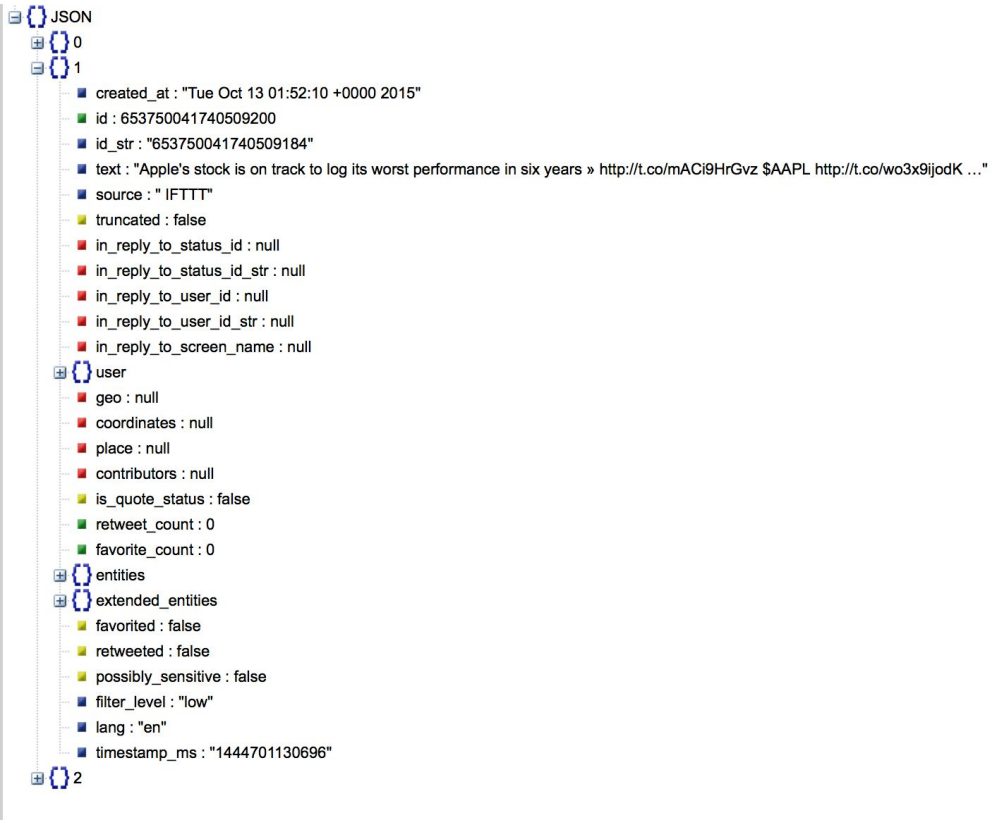
We would then use Spark Streaming which has helper utilities and example code for ingestion of tweets.

Spark Streaming Examples:

1. Twitter-based examples in Scala: https://github.com/apache/spark/tree/master/examples/src/main/scala/org/apache/spark/examples/streaming
2. General python spark streaming examples: https://github.com/apache/spark/tree/master/examples/src/main/python/streaming

The search API is more relevance based and could be used for grabbing historical data. From a data completeness point of view, Twitter recommends using the streaming API.

For example, using the streaming API and tracking a bunch of various stock tickers, (https://stream.twitter.com/1.1/statuses/filter.json?track=$BABA,$BIDU,$NYSE,$NASDAQ,$QQQ,$GOOG,$NFLX,$AAPL,$TSLA), the obtained json-based feed looks something like the below:



This data would be collected and written to HDFS in batches based on the period within which the twitter feed was obtained.

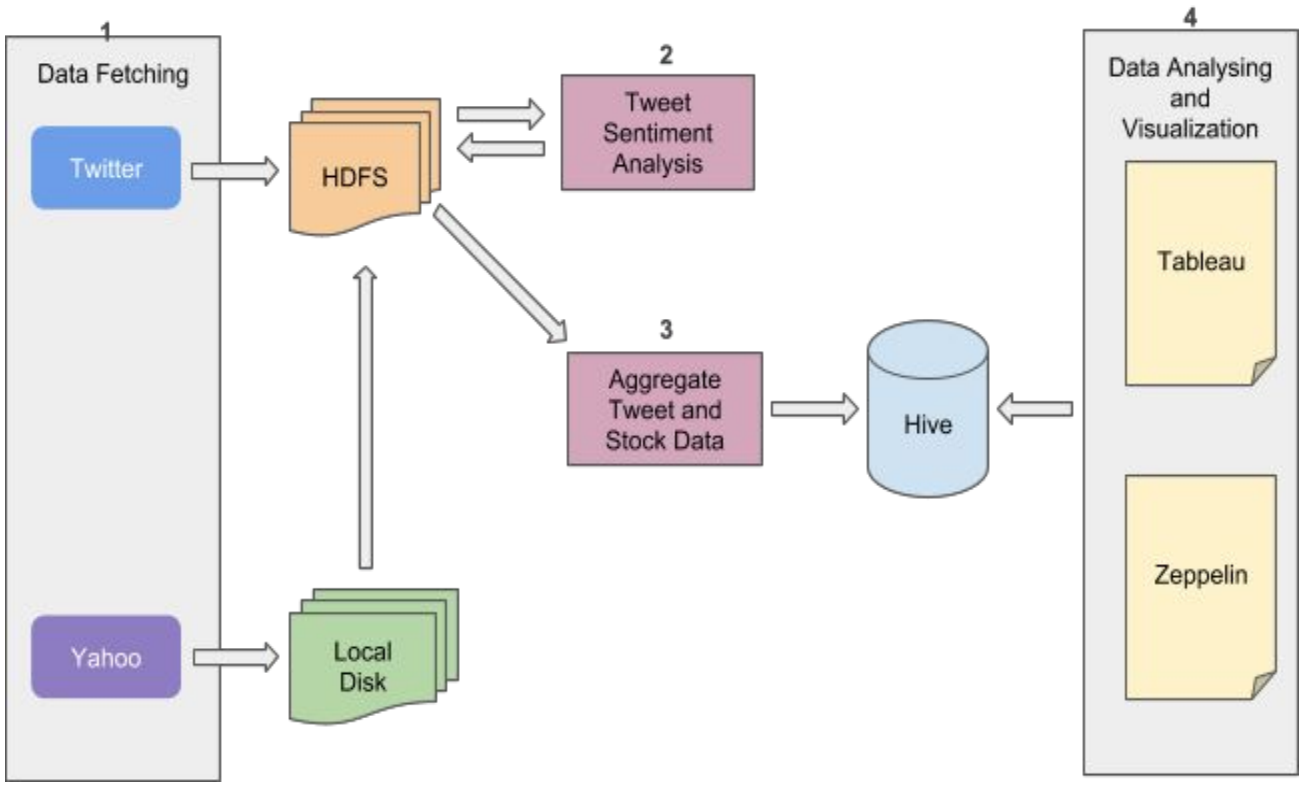
2.2. Yahoo Stock Data

For stock data, we decided to use a python wrapper to the Yahoo Finance Data YQL api (github repo: https://github.com/gurch101/StockScraper; documentation: http://www.gurchet-rai.net/dev/yahoo-finance-yql). YQL is a proprietary querying language developed by Yahoo's Developer Network. Although the syntax can be quite different, YQL is designed to work much like querying a regular RDBMS database, although what it's actually doing is making an API call to the database. Yahoo makes many of its data tables available to the public, and for our purposes, the ones that were relevant were the FINANCE_TABLES table and the CSV table, from Yahoo Finance.

The python wrapper that we used is called stockretriever.py in our github repo. It's a simple script that takes certain specific YQL queries and exposes methods to make the queries callable via python. Although using the python wrapper rather than just using YQL or the Yahoo API directly creates a lot of limitations in how we pulled the stock data, because our data requirements for stock data was relatively simple, we felt it made sense from a readability and collaboration perspective to use the python script.

Using the wrapper made it impossible to add aggregate operators, joins, and most importantly, where clauses on our queries, but at least for the first two limitations, those were features that we did not need to pull the stock data, so in the long run using the wrapper saved us considerable time. The limitation on the where clause was a blocker, but a tweak in the methods in the wrapper enabled us to get the data we needed. The two main features of this wrapper that we used were the get_current_info method for today's stock information, and the get_historical_info method for historical data. The get_current_info method is just the vanilla version from the original stockretriever.py file, but the get_historical_info method needed to be tweaked pretty heavily, because it did not allow us to set specific date parameters, which would only be possible if the where clause were somehow exposed in the method. The updated version of get_historical_info allowed for start date and end dates to be inputted.

3. System Architecture



3.1. Tools and Technologies used

Data Acquisition

- Twitter Spark streaming
- Yahoo finance data via python

Data Storage

- HDFS as data store for raw tweets and stock price on EC2
- Hive as data store for the aggregate table post processing of data

Processing

- Pyspark script to determine sentiment score of tweets using Indico and textblob
- Hive scripts to aggregate data

Reporting | UX

- Tableau connecting to aggregated Hive table
- R to do correlation analysis on the data extract from Hive table

Amazon EC2 Configuration

- **Data Processing Instance**
  Instance Type: m3.large
  CPU: 2 CPU's
  Memory: 7.5 GB
- **Data Storage:**
  EBS storage : 10 GB attached to Data processing Instance
- **AMI Used**
  ucbw205_complete_plus_postgres_PY2.7

4. Data Collection and Processing

4.1. Twitter Data

We are using spark streaming to get the tweets from Twitter. The code is written using Spark (Scala). Instead of listening to all tweets we have added a filter that listens to tweets that belong to the 30 stocks that make up the Dow Jones Industrial Average (DJIA). The streaming code takes as one of the input the filter file which contains the stock ticker symbols of the companies which makes up the DJIA e.g. $AAPL , $AXP,$BA,$CAT,$CSCO. A single tweet can at times contains multiple stock ticker symbols. Such tweets are converted into multiple records, one for each stock ticker symbol, to aid our data analysis which is based on each company (implemented using Spark dataframe's explode functionality). The data is stored in HDFS using a json-file format in the following directory structure:
<base_dir>/tweets/<date>/<twitter poll timestamp>/<actual tweet data files>
The data is not segregated on a per-stock basis as we could potentially change/increase the no. of companies being monitored later. For now, the data is partitioned by date to make it easier for data analysis against the stock prices for a given date range.

Below is the snapshot of the streaming code running.

```
Entering rdd loop: 940 Time = 1449718620000 ms
Number of tweets : 1
Entering rdd loop: 941 Time = 1449718640000 ms
Number of tweets : 14
Entering rdd loop: 942 Time = 1449718660000 ms
Number of tweets : 12
Entering rdd loop: 943 Time = 1449718680000 ms
Number of tweets : 1
Entering rdd loop: 944 Time = 1449718700000 ms
Number of tweets : 3
Entering rdd loop: 945 Time = 1449718720000 ms
Number of tweets : 12
```

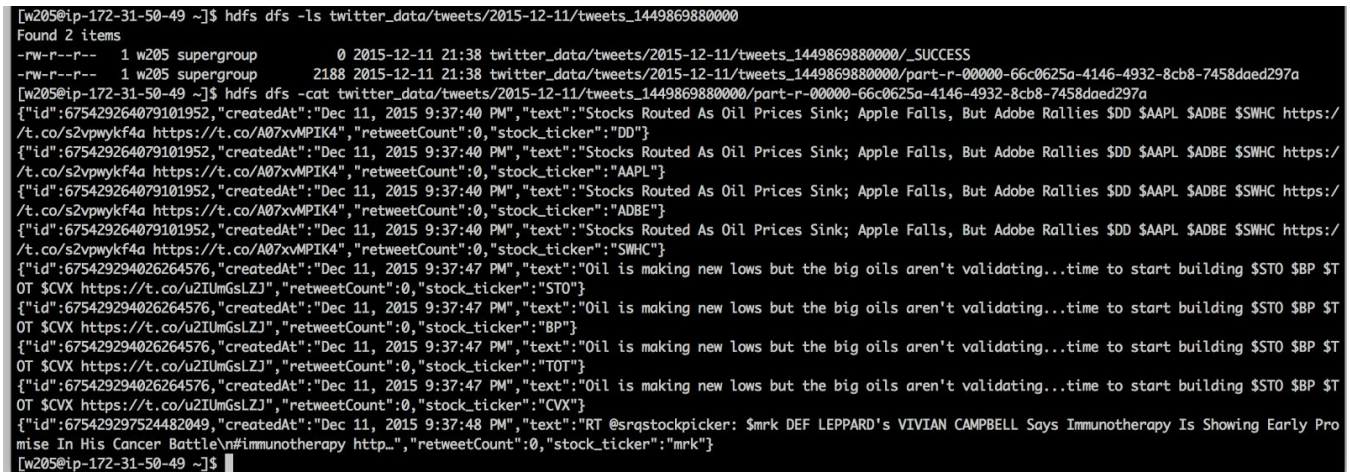
HDFS structure for storing raw tweets from twitter

```
[w205@ip-172-31-50-49 ~]$ hdfs dfs -ls twitter_data/tweets/
Found 29 items
drwxr-xr-x   - w205 supergroup          0 2015-11-15 21:20 twitter_data/tweets/2015-11-01
drwxr-xr-x   - w205 supergroup          0 2015-11-15 21:20 twitter_data/tweets/2015-11-03
drwxr-xr-x   - w205 supergroup          0 2015-11-15 22:04 twitter_data/tweets/2015-11-04
drwxr-xr-x   - w205 supergroup          0 2015-11-16 00:37 twitter_data/tweets/2015-11-05
drwxr-xr-x   - w205 supergroup          0 2015-11-16 02:05 twitter_data/tweets/2015-11-06
drwxr-xr-x   - w205 supergroup          0 2015-11-16 02:05 twitter_data/tweets/2015-11-07
drwxr-xr-x   - w205 supergroup          0 2015-11-16 02:05 twitter_data/tweets/2015-11-10
drwxr-xr-x   - w205 supergroup          0 2015-11-16 02:05 twitter_data/tweets/2015-11-11
drwxr-xr-x   - w205 supergroup          0 2015-11-16 02:06 twitter_data/tweets/2015-11-13
drwxr-xr-x   - w205 supergroup          0 2015-11-16 02:10 twitter_data/tweets/2015-11-14
drwxr-xr-x   - w205 supergroup          0 2015-11-16 02:11 twitter_data/tweets/2015-11-15
drwxr-xr-x   - w205 supergroup          0 2015-11-18 23:59 twitter_data/tweets/2015-11-18
drwxr-xr-x   - w205 supergroup          0 2015-11-19 04:23 twitter_data/tweets/2015-11-19
drwxr-xr-x   - w205 supergroup          0 2015-11-24 04:25 twitter_data/tweets/2015-11-24
drwxr-xr-x   - w205 supergroup          0 2015-11-25 23:59 twitter_data/tweets/2015-11-25
drwxr-xr-x   - w205 supergroup          0 2015-11-26 03:10 twitter_data/tweets/2015-11-26
drwxr-xr-x   - w205 supergroup          0 2015-11-27 23:59 twitter_data/tweets/2015-11-27
drwxr-xr-x   - w205 supergroup          0 2015-11-28 01:40 twitter_data/tweets/2015-11-28
drwxr-xr-x   - w205 supergroup          0 2015-12-01 05:25 twitter_data/tweets/2015-12-1
drwxr-xr-x   - w205 supergroup          0 2015-12-10 22:37 twitter_data/tweets/2015-12-10
drwxr-xr-x   - w205 supergroup          0 2015-12-11 21:39 twitter_data/tweets/2015-12-11
drwxr-xr-x   - w205 supergroup          0 2015-12-02 23:59 twitter_data/tweets/2015-12-2
drwxr-xr-x   - w205 supergroup          0 2015-12-03 04:49 twitter_data/tweets/2015-12-3
drwxr-xr-x   - w205 supergroup          0 2015-12-04 23:59 twitter_data/tweets/2015-12-4
drwxr-xr-x   - w205 supergroup          0 2015-12-05 23:59 twitter_data/tweets/2015-12-5
drwxr-xr-x   - w205 supergroup          0 2015-12-06 23:58 twitter_data/tweets/2015-12-6
drwxr-xr-x   - w205 supergroup          0 2015-12-07 00:44 twitter_data/tweets/2015-12-7
drwxr-xr-x   - w205 supergroup          0 2015-12-08 05:26 twitter_data/tweets/2015-12-8
drwxr-xr-x   - w205 supergroup          0 2015-12-09 23:59 twitter_data/tweets/2015-12-9
```

Instead of storing the entire tweet, which contains information we don't need, the streaming code only stores the tweet id, created datetime, tweet text, retweet count and stock ticker symbol from the tweet. This drastically reduces the storage required per tweet to roughly ± 254 Bytes. The spark streaming code currently has an upper limit of 5000 tweets after which we exit the program. This will mainly done to avoid being throttled/blocked by Twitter and the code can easily be changed to increase this limit if needed. Running it daily, we haven't hit that limit yet because we are filtering out the tweets based only on the stock ticker symbol v/s hashtags. Below is a sample json tweet that we are storing:

{"id":665764640677228544,"createdAt":"Nov 15, 2015 5:33:54 AM","text":"RT @jsonyoung: GE's stock will move 1.5% (on average) each day over the next month if the options market is right: https://t.co/IWfNGU9aXb/…","retweetCount":0,"stock_ticker":"GE"}.

Snapshot of data stored in hdfs from one pull of data from twitter.



We are storing the data in HDFS and a sample run for an entire day resulted in approximately 1.1 MB of processed data. Velocity is in the single digits i.e. we poll twitter every 20 seconds and the filtered tweet count is usually under 10. There are some outliers where we get 10s of tweets but never over 100. To summarise on a per minute basis we seem to be averaging between 10 to 20 tweets per minute that we process.

From a performance point of view, for a streaming application to not lag, the data needs to be processed before the next batch is available. Based on initial runs on an m3.large EC2 instance, the general processing time for a dozen tweets was less than 10 seconds. Hence to be safe (and also as the number of tweets to process is usually under 10) the polling interval is set to 20 seconds. The larger interval is mostly to ensure that app is not blocked by twitter. (Note: On a m3.medium instance to use spark streaming you need to set the following parameter " --master local[3]" as it does not pick up the correct number of worker threads.)

4.2. Yahoo Stock Data

In order to collect the stock data we wrote a script that uses stockretriever.py to output json files to a specified directory (eventually to HDFS for the purposes of this project). It also references stock_symbols, which is an array store of stock symbols of the Dow Jones 30 (the S&P 100 stock symbols are in there as well, but we decided to go with Dow Jones):

A screenshot of the stock_symbols.py file, containing S&P 100 stock symbols and Dow Jones 30 stock symbols.

Both stockretriever.py and stock_symbols.py are imported into stock_pusher.py, which, when run, provide the ability to get either the day's stock information or any historical data. It pulls all stocks from the Dow Jones 30 every time it's run, stores the data in a user-specified directory.

The bulk of the work that stock_pusher.py does is calling the Yahoo Finance API and processing the data to make it fit the format needed for analysis. When the API is called using the methods exposed in stockretriever.py, these are the data files we get for the day's data and the historical data, respectively:

data pulled for the day:

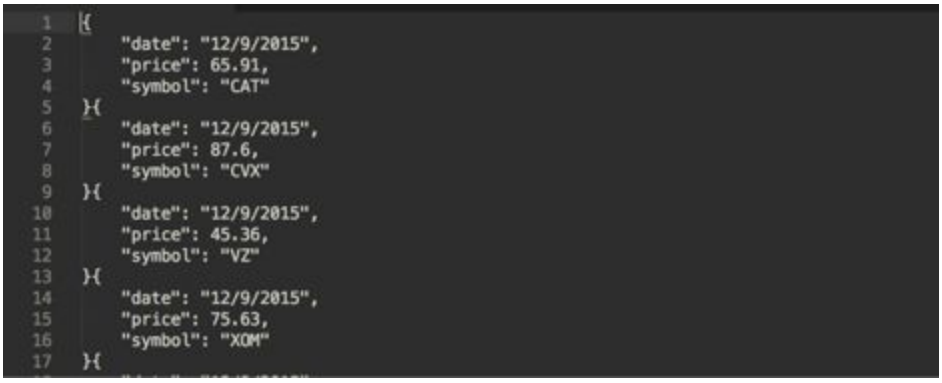
A screenshot of the day's raw stock data.

historical data (pulled for 11/01/15-11/30/15):

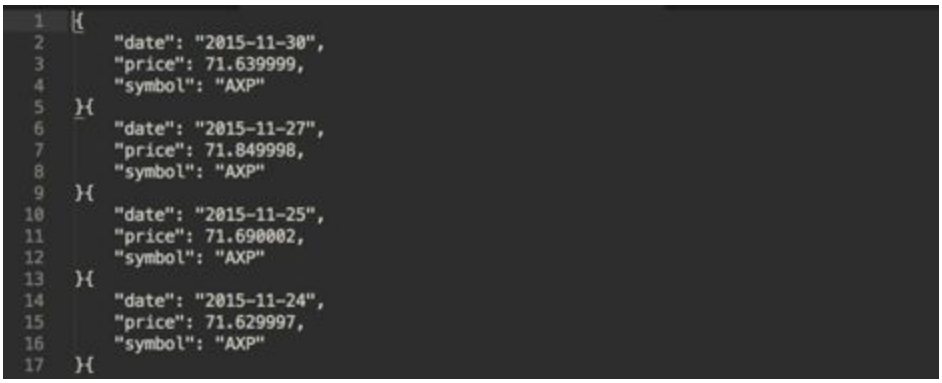
A screenshot of the raw historical data (11/1/15-11/30/15)

For each stock data event pulled, there are 83 variables when the “today” option is used, and 7 variables when the “historical” option is used. For the purposes of this project, all we really need are three variables: Symbol, Date, and Price. So stockretriever.py parses out the relevant variables and restructures the data into json-like text files so that it will be easily ingested into HDFS, then a Hive table for easy querying when doing analysis. Here are the outputs:

For the day’s data:



For historical data:



5. Analysis

As discussed previously, the project has 2 main sources of data:
Twitter data: The twitter data stream fetches relevant tweets about the 30 companies included in the Dow Jones Industrial average(DJIA).
Yahoo stock data: Calls to the Yahoo Finance api retries daily closing prices for the 30 DJIA companies.

Ultimately, we plot the average sentiment score(normalized for a daily value) for the company over a specified number of days and analyse how that trends in correlation to the stock price and the count of tweets. The count of tweets would represent a naive interpretation of how popular the company was on a given day in the twitter world while the sentiment score metric would bring context to this number by incorporating the positive or negative sentiment of the tweets.

To accomplish this, we start off by calculating the sentiment scores of the individual tweets. This is accomplished by calls to the Indico sentiment score API and the text-blob sentiment api.
We found these to elicit more reasonable sentiment scores for the tweets in comparison to the Sentiment140 API we had initially planned to use to.

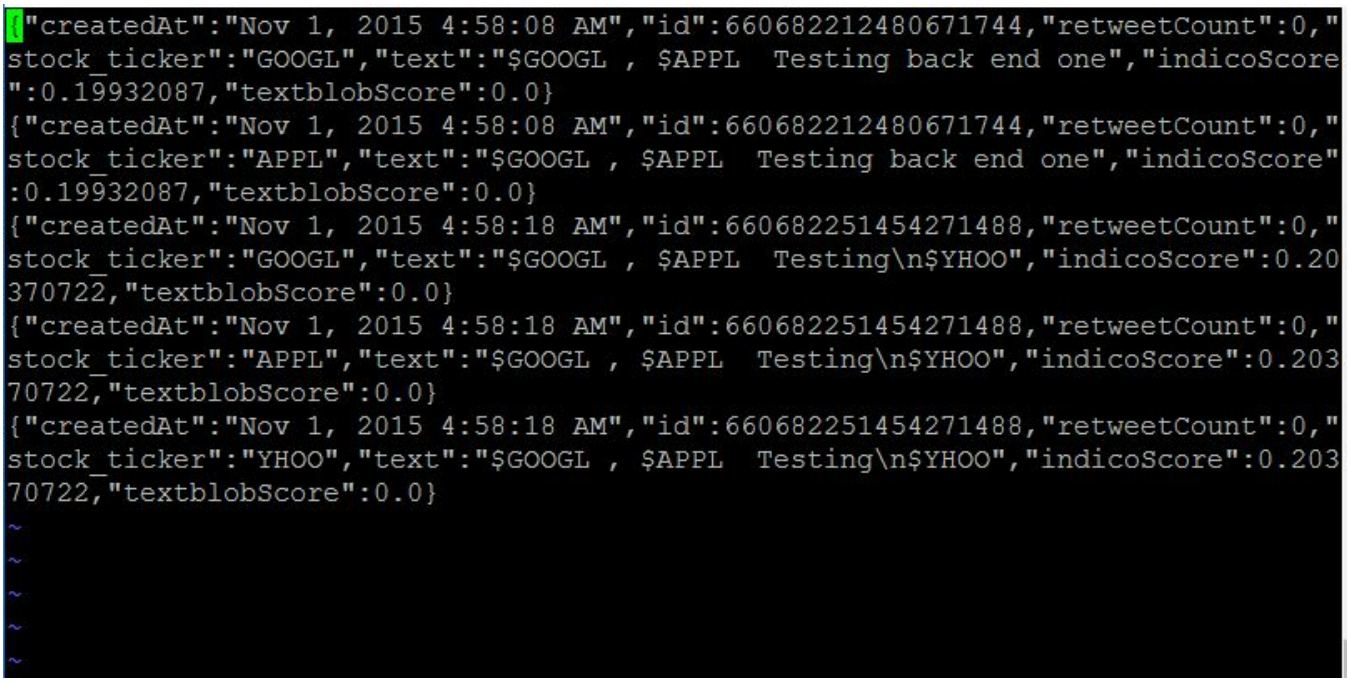
5.1. Sentiment Analysis

The sentiment analysis code is again written in spark(pyspark) and processes data that is being written in hdfs from the streaming code. The Indico API returns a number between 0 and 1. This number is a probability representing the likelihood that the analyzed text is positive or negative. Textblob is available as a python package. The sentiment property returns a polarity score within the range -1.0 and 1.0. The data is stored in HDFS using a json-file format in the following directory structure:

twitter_data/tweets/processed/<model name>/<date>/.

This structure allowed for being able to apply different sentiment analysis models to the data and storing them in different locations. The code has option to run each model individually or do a combined code analysis. The approach in the code is to currently use a special case model named “combined” which applies all available models to the raw data.

Textblob offers two different methods for sentiment analysis : Pattern analyzer and Naive Bayes analyzer. Currently the combined model does not run the text blob naive bayes analyzer as its performance is too slow.


{"createdAt":"Nov 1, 2015 4:58:08 AM","id":660682212480671744,"retweetCount":0,"
stock_ticker":"GOOGL","text":"$GOOGL , $APPL Testing back end one","indicoScore
":0.19932087,"textblobScore":0.0}
{"createdAt":"Nov 1, 2015 4:58:08 AM","id":660682212480671744,"retweetCount":0,"
stock_ticker":"APPL","text":"$GOOGL , $APPL Testing back end one","indicoScore"
:0.19932087,"textblobScore":0.0}
{"createdAt":"Nov 1, 2015 4:58:18 AM","id":660682251454271488,"retweetCount":0,"
stock_ticker":"GOOGL","text":"$GOOGL , $APPL Testing\n$YHOO","indicoScore":0.20
370722,"textblobScore":0.0}
{"createdAt":"Nov 1, 2015 4:58:18 AM","id":660682251454271488,"retweetCount":0,"
stock_ticker":"APPL","text":"$GOOGL , $APPL Testing\n$YHOO","indicoScore":0.203
70722,"textblobScore":0.0}
{"createdAt":"Nov 1, 2015 4:58:18 AM","id":660682251454271488,"retweetCount":0,"
stock_ticker":"YHOO","text":"$GOOGL , $APPL Testing\n$YHOO","indicoScore":0.203
70722,"textblobScore":0.0}

5.2. Data Aggregation

We then load the raw twitter JSON data for all days from HDFS into a hive table. This table is called raw_tweets. From raw_tweets we select the needed fields: stock_ticker, date, indico sentiment score and textblob sentiment score and write those into another table: tweets in the needed format.

```
hive> select * from tweets LIMIT 20
    > ;
OK
MAT     2015-11-01      0.45896402      0.0
PAY     2015-11-01      0.45896402      0.0
FB      2015-11-01      0.45896402      0.0
GOOGL   2015-11-01      0.23979431      0.0
APPL    2015-11-01      0.23979431      0.0
GOOGL   2015-11-01      0.07865393      0.0
APPL    2015-11-01      0.07865393      0.0
TSLA    2015-11-01      0.07865393      0.0
GOOGL   2015-11-01      0.19932087      0.0
APPL    2015-11-01      0.19932087      0.0
GOOGL   2015-11-01      0.20370722      0.0
APPL    2015-11-01      0.20370722      0.0
YHOO    2015-11-01      0.20370722      0.0
XOM     2015-11-01      0.5860208       0.0
AXP     2015-11-01      0.59319603      0.0
SGRY    2015-11-01      0.59319603      0.0
BCOM    2015-11-01      0.59319603      0.0
TVK.TO  2015-11-01      0.59319603      0.0
BR      2015-11-01      0.12889458      0.0
AMGN    2015-11-01      0.12889458      0.0
```

The table is then grouped by the stock_ticker(company) and the date column to have an aggregate table(tweet_data_per_day) with the avg indico and textBlob sentiment scores and the count of tweets at the company, date level.

```
hive> select * from tweet_data_per_day LIMIT 30;
OK
AAPL    2015-11-06      979     0.502266934540858       0.07120742812737477
DIS     2015-11-06      427     0.39652975764168635     0.045070506653395806
GS      2015-11-06      341     0.37746456737243383     0.07148890442815241
MSFT    2015-11-06      278     0.4736747057230213      0.12324792122014393
MRK     2015-11-06      216     0.32676081799768525     0.028709865902777774
INTC    2015-11-06      206     0.3517815145888347      0.020545052281553393
XOM     2015-11-06      191     0.5268595740366493      0.04169014406282722
IBM     2015-11-06      184     0.5370061705054346      0.05731363509782616
JPM     2015-11-06      177     0.31854521033050837     0.04523340681920904
CSCO    2015-11-06      152     0.498828721236842       0.07136975940789474
GE      2015-11-06      149     0.4504354958020132      0.04620936912080537
CVX     2015-11-06      134     0.33707933319402983     -0.006036672686567164
JNJ     2015-11-06      126     0.4705321300396826      0.14264349865079365
V       2015-11-06      123     0.6443149742357721      0.06661759869918699
VZ      2015-11-06      114     0.3772339067807016      0.05309897228070175
PFE     2015-11-06      106     0.5873237864245281      -0.17854702681132067
WMT     2015-11-06      104     0.4882025152019232      0.04109715140096153
CAT     2015-11-06      92      0.5473308650543479      -0.0043480414673913075
NKE     2015-11-06      91      0.46898459747252735     0.0329888628978022
MCD     2015-11-06      67      0.4586880961791045      0.019323916716417908
DD      2015-11-06      48      0.4587248356041667      0.04457702020833334
MMM     2015-11-06      44      0.45564745749999974     0.12027376040909094
KO      2015-11-06      34      0.564701502764706       0.11261533352941176
BA      2015-11-06      34      0.42657627138235304     0.05371716064705882
HD      2015-11-06      32      0.50999287690625        0.037762295803125
PG      2015-11-06      28      0.46898936460714286     0.08168792500000001
AXP     2015-11-06      15      0.4897668966666666      0.10052777800000001
UNH     2015-11-06      9       0.38674210733333336     0.05555555555555555
AAPL    2015-11-03      4       0.5446539075000001      0.10291666000000002
AAPL    2015-11-01      3       0.7312067133333334      0.0
Time taken: 0.088 seconds, Fetched: 30 row(s)
```

In a similar fashion, the raw stock JSON data gets pulled into a raw table(raw_stock_prices). Form here it gets loaded into the table stock_prices with the formatted fields we need: stock ticker, date and stock price.

```
hive> select * from stock_prices;
OK
AAPL    2015-11-06      121.059998
AXP     2015-11-06      74.300003
BA      2015-11-06      147.940002
CAT     2015-11-06      73.839996
CSCO    2015-11-06      28.450001
CVX     2015-11-06      94.029999
DD      2015-11-06      66.110001
DIS     2015-11-06      115.669998
GE      2015-11-06      29.92
GS      2015-11-06      199.169998
HD      2015-11-06      125.980003
IBM     2015-11-06      138.25
INTC    2015-11-06      33.84
JNJ     2015-11-06      101.919998
JPM     2015-11-06      68.459999
KO      2015-11-06      41.959999
MCD     2015-11-06      113.309998
MMM     2015-11-06      159.259995
MRK     2015-11-06      54.610001
MSFT    2015-11-06      54.919998
NKE     2015-11-06      131.779999
PFE     2015-11-06      33.93
PG      2015-11-06      75.57
TRV     2015-11-06      113.099998
UNH     2015-11-06      114.809998
UTX     2015-11-06      100.800003
VZ      2015-11-06      45.779999
V       2015-11-06      78.75
WMT     2015-11-06      58.779999
XOM     2015-11-06      84.470001
Time taken: 0.086 seconds, Fetched: 30 row(s)
```

Next the aggregate tweet_data_per_day and stock_prices tables are joined on the stock ticker and date fields to produce the final table with the following fields: stock_ticker, date, tweet count, indico sentiment score, textblob sentiment score and the stock price. This is the final table of our analysis from which the visualization works.

```
hive> select * from stock_tweet_data limit 20;
OK
AAPL    2015-11-06      979     0.502266934540858       0.07120742812737477     121.059998
DIS     2015-11-06      427     0.39652975764168635     0.045070506653395806    115.669998
GS      2015-11-06      341     0.37746456737243383     0.07148890442815241     199.169998
MSFT    2015-11-06      278     0.4736747057230213      0.12324792122014393     54.919998
MRK     2015-11-06      216     0.326760817997685 25    0.028709865902777774    54.610001
INTC    2015-11-06      206     0.3517815145888347      0.020545052281553393    33.84
XOM     2015-11-06      191     0.5268595740366493      0.04169014406282722     84.470001
IBM     2015-11-06      184     0.5370061705054346      0.05731363509782616     138.25
JPM     2015-11-06      177     0.31854521033050837     0.04523340681920904     68.459999
CSCO    2015-11-06      152     0.498828721236842       0.07136975940789474     28.450001
GE      2015-11-06      149     0.4504354958020132      0.04620936912080537     29.92
CVX     2015-11-06      134     0.33707933319402983     -0.006036672686567164   94.029999
JNJ     2015-11-06      126     0.4705321300396826      0.14264349865079365     101.919998
V       2015-11-06      123     0.6443149742357721      0.06661759869918699     78.75
VZ      2015-11-06      114     0.3772339067807016      0.05309897228070175     45.779999
PFE     2015-11-06      106     0.5873237864245281      -0.17854702681132067    33.93
WMT     2015-11-06      104     0.4882025152019232      0.04109715140096153     58.779999
CAT     2015-11-06      92      0.5473308650543479      -0.0043480414673913075  73.839996
NKE     2015-11-06      91      0.46898459747252735     0.0329888628978022      131.779999
MCD     2015-11-06      67      0.4586880961791045      0.019323916716417908    113.309998
Time taken: 0.14 seconds, Fetched: 20 row(s)
```

6. Visualization

We use Tableau for our visualization wherein the user will be able to select from the list of companies that constitute the DJIA from a drop down selector and that will show the how the stock prices and twitter sentiment trend against each other during the relevant time frame.

Tableau queries against the final analysis table and for the 30 day time frame, we plot the sentiment, tweet count and stock price for the given company.

From the graphs above it might look like there is some correlation between stock price , tweet count and sentiment score. We did some analysis using R and below are the results of our test.

1. Running correlation across the entire data set. Based on the results below, there is no correlation between either price and tweetcount or price and sentiment score.

> cor(stock_tweet[,c("Price","Tweetcount","Indicoscore","Textblobscore")])

	Price	Tweetcount	Indicoscore	Textblobscore
Price	1.000000000	0.07554435	0.03609364	-0.009150521

Tweetcount	0.075544348	1.000000000	-0.03276538	0.073710680
Indicoscore	0.036093639	-0.03276538	1.000000000	0.085251543
Textblobscore	-0.009150521	0.07371068	0.08525154	1.000000000

2. Running correlation for one company also showed us the same results.

	Price	Tweetcount	Indicoscore	Textblobscore
Price	1.000000000	0.27079769	0.1373933	0.10046971
Tweetcount	0.2707977	1.000000000	0.1822657	0.06056054
Indicoscore	0.1373933	0.18226568	1.000000000	0.64869892
Textblobscore	0.1004697	0.06056054	0.6486989	1.000000000

7. Lessons Learned

Implementing this project has been quite helpful in reinforcing concepts that were introduced in class. Among the many lessons learned few have been highlighted below:

a) The tweets that we are writing to HDFS are very small in size. Instead of writing to data to HDFS we could have written data to s3, nfs mounted file system or any reliable data source.

b) We spent quite some time trying to transform the JSON data format before loading it to hive tables using the glob package. Eventually we figured out that creating hive tables and loading the raw tweet and stock data into the tables and later transforming the data was much was straight-forward.

c) Most of the processing currently is in batch mode. We capture the data on a daily basis and do post processing. We could potentially do this in more real time i.e. some form of streaming pipeline which can give trends within a matter of seconds / minutes.

8. Future Roadmap

a) We chose to use text blob and indico sentiment score API's to calculate the positive/negative sentiment scores for the tweets. Though the sentiment scores seemed fairly reasonable, implementing a custom classifier would help improve the model reliability further.
b) For our implementation, we chose to filter out the tweets using the company stock ticker symbol. improvements would see us expanding the twitter streaming filter such we are looking for all mentions of the company (different flavour of the company name included using stemming techniques)
c) Use other sources of data for social media such as news, facebook etc.

9. Source Code and Data

Github link：https://github.com/kyungguyeo/205_project

Data : https://github.com/kyungguyeo/205_project/tree/master/data