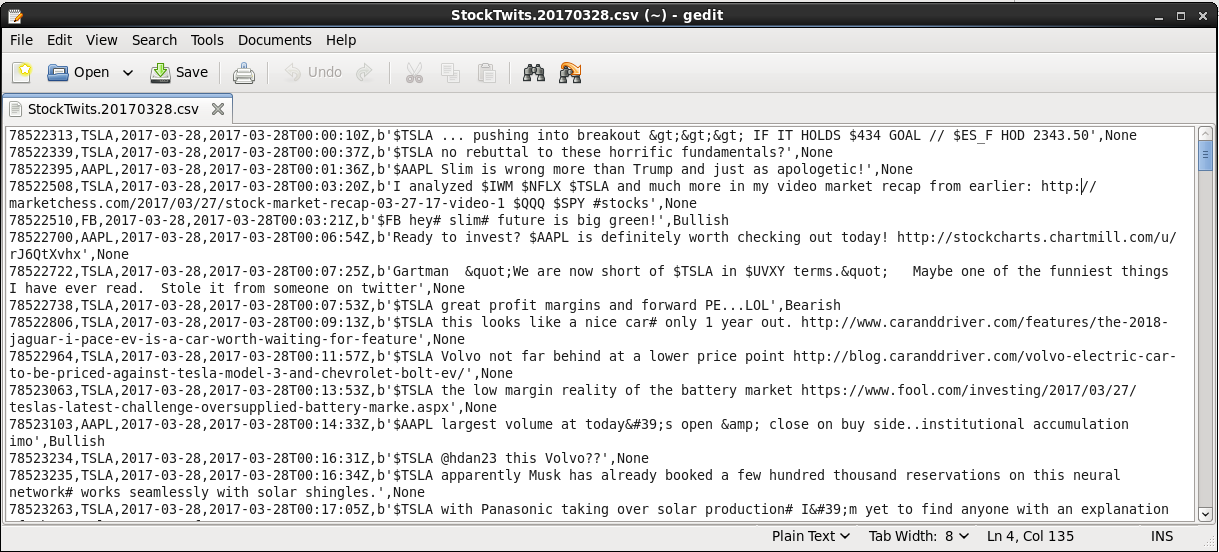
**CSCI63 Final Project Report: NLTK and Spark Near-Real Time Text Analysis**

**Spark Streaming**

The goal of this section is to utilize Spark streaming technology to demonstrate real-time parallel processing of data. To illustrate near-real time Spark streaming, we use the provided files stream-dict.py and splitAndSendFinal.sh. For data, we use a csv file which contained all StockTwits downloaded from March 28th, 2017. The file (StockTwits.20170328.csv) is sorted chronologically and does not include any row headers. It is in the same column format as the StockTwits data files we had downloaded. This file was copied over to the Cloudera home directory /home/cloudera/. In addition, I copied over the two dictionary files LoughranMcDonald\_MasterDictionary\_2014.xlsx and inquirerbasic.xls into the Cloudera home directory.

*StockTwits.20170328.csv*



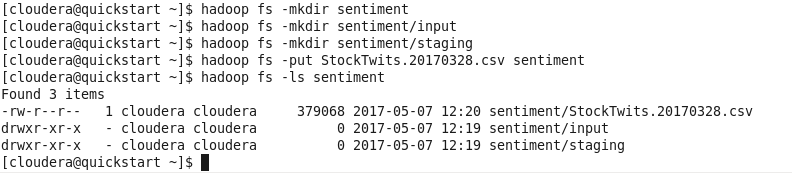
We run the following commands to set up the file directory and copy necessary files over:

*hadoop fs –mkdir sentiment*

*hadoop fs –mkdir sentiment/input*

*hadoop fs –mkdir sentiment/staging*

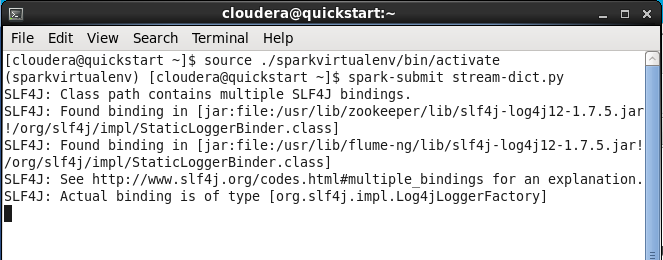
*hadoop fs –put StockTwits.20170328.csv sentiment*

**

In one terminal window, we run the Spark stream listener which runs the sentiment analysis using the Harvard dictionary on a Bag-Of-Words. We can run the process in a new terminal window:

*source ./sparkvirtualenv/bin/activate*

*spark-submit stream-dict.py*

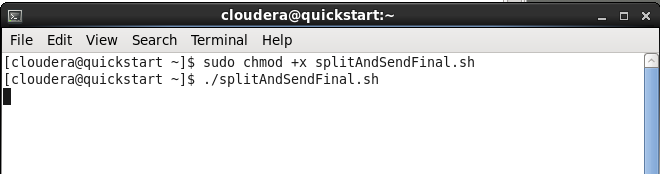
**

In a different terminal, we need to change permissioning to run the splitAndSendFinal script and run it:

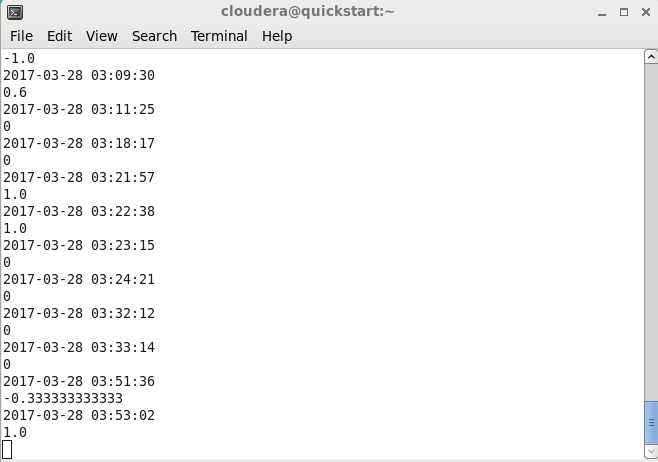
*source ./sparkvirtualenv/bin/activate*

*sudo chmod +x splitAndSendFinal.sh*

*./splitAndSendFinal.sh*

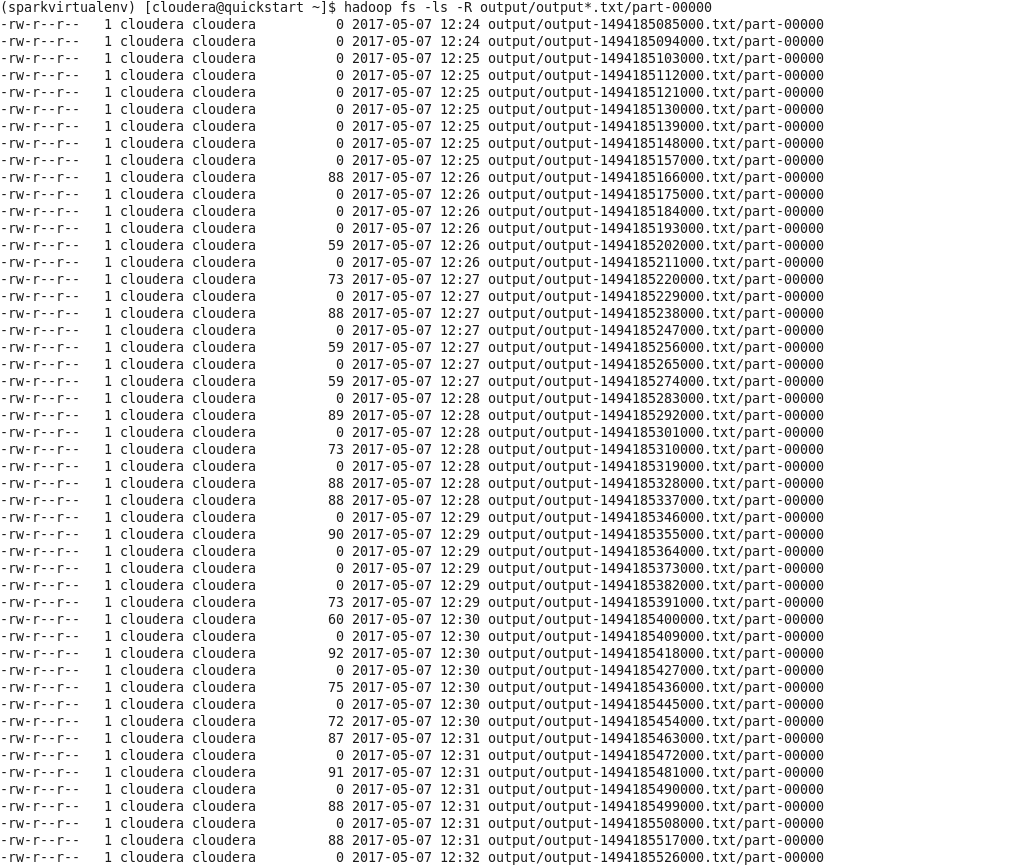
**

The splitAndSendFinal file divides tweets into clusters of 100 tweets, and pauses 10 seconds before sending another batch to the listener.



Our listener (stream-dict.py) provides some intermediate information in the terminal by displaying the immediate score for each single tweet. There were 2,791 tweets for the three stocks on March 28th, the sender streamed data through in 28 chunks, and everything finished in roughly 5 minutes.

To view the result files: *hadoop fs –ls –R output/output\*.txt/part-00000*

**

To view individual results:

*hadoop fs -cat output/output-1494185517000.txt/part-00000*

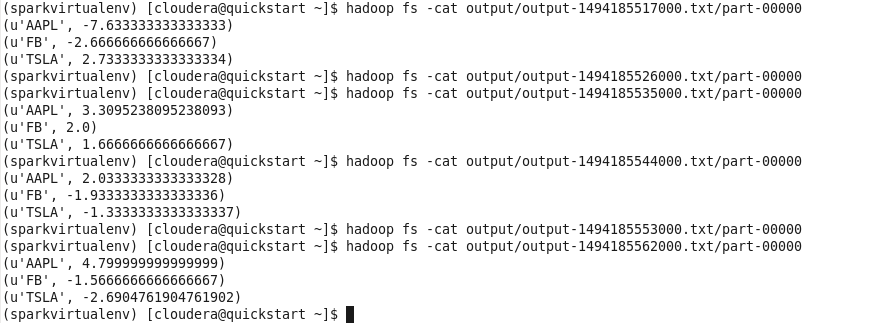
*hadoop fs -cat output/output-1494185526000.txt/part-00000*

*hadoop fs -cat output/output-1494185535000.txt/part-00000*

*hadoop fs -cat output/output-1494185544000.txt/part-00000*

*hadoop fs -cat output/output-1494185553000.txt/part-00000*

*hadoop fs -cat output/output-1494185562000.txt/part-00000*

**

The final output provides a total sentiment score for each stock during that time interval it was received. In real-time streaming, we calculate the raw sum of sentiment scores rather than total sentiment score divided by the number of tweets, because the distribution of tweets is fairly random. Also, notice that some of the output files were empty. This is due to the 10-second pause in between simulated tweets being broadcast.

If we wanted to clear the output files and re-run:

*hadoop fs –rm output/output\*.txt/\**

*hadoop fs –rmdir output/\**

Below is a chart of our final output using the Harvard dictionary for all three stocks. For each of the 28 data chunks that were released, the Spark program generated an output file which summarized the total sentiment for each of the three stock. The data was then collected and plotted. If you refer back to the earlier chart with intraday analysis of AAPL on March 28th plotted against price, the blue line here matches up almost exactly with the blue line (as it should!) on that chart which denotes Sentiment (Harvard). The only minor difference was that the earlier chart was done on a trading day and, this was done over the entire 24-hour period.



The advantage of applying real-time streaming to a historical data we already tested is that we can check that the results were in line with what we expected. In addition, Spark was is able to process our test case nearly instantaneously due to simple nature of the dictionary lookup in our Bag-Of-Words model. If we had run a much more complex model or run in shorter time intervals, it would likely not have had issues either because of Spark’s parallel processing capabilities.