Homework 1

1. Download and Install Spark. Learn how to use it.

I first started Spark using the command: ./bin/pyspark.

```
bigdata@ubuntu:/usr/local/spark-1.6.0-bin-hadoop2.4$ ./bin/pyspark
Python 2.7.6 (default, Jun 22 2015, 17:58:13)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
16/02/18 13:01:24 INFO SparkContext: Running Spark version 1.6.0
```

I then tested out commands in Spark (lines = sc.textFile("README.md"), lines.count())

```
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 1.6.0
      /_/

Using Python version 2.7.6 (default, Jun 22 2015 17:58:13)
SparkContext available as sc, HiveContext available as sqlContext.
>>> lines = sc.textFile("README.md")
16/02/18 13:01:57 INFO MemoryStore: Block broadcast_0 stored as values in memory
 (estimated size 140.5 KB, free 140.5 KB)
16/02/18 13:01:57 INFO MemoryStore: Block broadcast_0_piece0 stored as bytes in
memory (estimated size 12.4 KB, free 152.8 KB)
16/02/18 13:01:57 INFO BlockManagerInfo: Added broadcast_0_piece0 in memory on l
ocalhost:40881 (size: 12.4 KB, free: 517.4 MB)
16/02/18 13:01:57 INFO SparkContext: Created broadcast 0 from textFile at Native
MethodAccessorImpl.java:-2
>>> lines.count()
16/02/18 13:02:03 INFO FileInputFormat: Total input paths to process : 1
16/02/18 13:02:03 INFO SparkContext: Starting job: count at <stdin>:1
```

Below is the output of lines.count(). We can see there are 95 lines in the README.md file.

```
16/02/18 13:02:11 INFO DAGScheduler: ResultStage 0 (count at <stdin>:1) finished
 in 8.199 s
16/02/18 13:02:11 INFO DAGScheduler: Job 0 finished: count at <stdin>:1, took 8.
653743 s
95
```

2. Download Wikipedia dataset. Extract about 100 pages (items) based on your own interest. You may use snowball method to crawl a few related/linked pages. Create TF-IDF of each page.

I first downloaded the Wikipedia dataset from the website. I downloaded a data dump of Wikipedia articles.

```
bigdata@ubuntu:~/AdvBigData/Homework1/Wikipedia$ ls
enwiki-latest-pages-articles1.xml-p000000010p000030302
```

Anubha Bhargava
Advanced Big Data Analytics

I extracted the Wikipedia data from the downloaded file using the script shown below:

```
import lxml.etree
import os
parseddata = lxml.etree.parse('/home/bigdata/AdvBigData/Homework1/Wikipedia/enwiki-latest-pages-articles1.
xml-p000000010p000030302')
ns = {'ns':'http://www.mediawiki.org/xml/export-0.10/'}
i = 0
list1 = tree.xpath('//ns:page', ns = ns)
for l in list1:
    txt = l.xpath('.//ns:title', ns = ns)[0].text
    print txt
    f = file('parse/' + str(i), 'w')
    txt1 = l.xpath('.//ns:revision/ns:text', ns = ns)[0].text
    f.write(txt.encode('utf-8') + '\n')
    f.write(txt1.encode('utf-8'))
    f.close()
    i = i + 1
```

The results of the Wikipedia extraction script are displayed:

```
11091  1221   13329  14448  15567  16686  17804  18923  2149   3268  4387  5505  6624  7743  8862  9981
11092  12210  1333   14449  15568  16687  17805  18924  215    3269  4388  5506  6625  7744  8863  9982
11093  12211  13330  1445   15569  16688  17806  18925  2150   327   4389  5507  6626  7745  8864  9983
11094  12212  13331  14450  1557   16689  17807  18926  2151   3270  439   5508  6627  7746  8865  9984
11095  12213  13332  14451  15570  1669   17808  18927  2152   3271  4390  5509  6628  7747  8866  9985
11096  12214  13333  14452  15571  16690  17809  18928  2153   3272  4391  551   6629  7748  8867  9986
11097  12215  13334  14453  15572  16691  1781   18929  2154   3273  4392  5510  663   7749  8868  9987
11098  12216  13335  14454  15573  16692  17810  1893   2155   3274  4393  5511  6630  775   8869  9988
11099  12217  13336  14455  15574  16693  17811  18930  2156   3275  4394  5512  6631  7750  887   9989
111    12218  13337  14456  15575  16694  17812  18931  2157   3276  4395  5513  6632  7751  8870  999
1110   12219  13338  14457  15576  16695  17813  18932  2158   3277  4396  5514  6633  7752  8871  9990
```

Then, I selected 100 pages of the Wikipedia extraction data:

```
bigdata@ubuntu:~/AdvBigData/Homework1/parse2$ ls
1      13003  1303   13090  13100  13131  13193  13303  1333   13390  13900  13913  13939
13     13009  13030  13091  13101  13133  13199  13309  13330  13391  13901  13919  1399
130    1301   13031  13093  13103  13139  133    1331   13331  13393  13903  1393   13990
1300   13010  13033  13099  13109  1319   1330   13310  13333  13399  13909  13930  13991
13000  13013  13039  131    1313   13190  13300  13313  13339  139    1391   13931  13993
13001  13019  1309   1310   13130  13191  13301  13319  1339   1390   13910  13933  13999
```

I then conducted TF-IDF using the following script, which grabs the data from the extracted data shown above. It outputs the TF data in a folder called TF_Wiki and the TF-IDF data in a folder called tfIdfVectors. TF counts the number of times a word appears in the document. The inverse document frequency (IDF) will determine which words are good indicators (unlike words like “the” and “so”) and

highlight those.

```
from pyspark.mllib.feature import HashingTF, IDF
from pyspark.mllib.clustering import KMeans, KMeansModel
from numpy import array
from pyspark import SparkContext
from math import sqrt

sc=SparkContext()
rdd=sc.wholeTextFiles("parse2/").map(lambda (name,text): text.split())
tf=HashingTF()
vectors=tf.transform(rdd).cache()
a = vectors.collect()
c = 0
for v in a:
        c = c + 1
        with open("TF_Wiki/tf_wiki"+str(c)+".txt","w") as f:
                f.write(str(v))
        f.close()
idf=IDF()
idfresult=idf.fit(vectors)
tfIdfVec=idfresult.transform(vectors)
tfIdfVec.saveAsTextFile("tfIdfVectors/")
~
```

```
bigdata@ubuntu:/usr/local/spark-1.6.0-bin-hadoop2.4$ ./bin/spark-submit /home/bigdata/AdvBigData/Homework
/tfidf.py
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
16/02/18 13:22:05 INFO SparkContext: Running Spark version 1.6.0
16/02/18 13:22:05 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using
uiltin-java classes where applicable
16/02/18 13:22:06 WARN Utils: Your hostname, ubuntu resolves to a loopback address: 127.0.0.1; using 192.
68.145.139 instead (on interface eth0)
16/02/18 13:22:06 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
```

If we go to the TF_Wiki folder and open one of the tf_wiki files, we get the following output. The output is when TF is conducted on one of the articles within the Wikipedia file. The following is for tf_wiki100.txt. It shows us the numbers corresponding to the words that occur the most.

```
bigdata@ubuntu:~/AdvBigData/Homework1/TF_Wiki$ ls
tf_wiki100.txt  tf_wiki14.txt  tf_wiki29.txt  tf_wiki43.txt  tf_wiki58.txt  tf_wiki72.txt  tf_wiki87.txt
tf_wiki101.txt  tf_wiki15.txt  tf_wiki2.txt   tf_wiki44.txt  tf_wiki59.txt  tf_wiki73.txt  tf_wiki88.txt
tf_wiki102.txt  tf_wiki16.txt  tf_wiki30.txt  tf_wiki45.txt  tf_wiki5.txt   tf_wiki74.txt  tf_wiki89.txt
tf_wiki103.txt  tf_wiki17.txt  tf_wiki31.txt  tf_wiki46.txt  tf_wiki60.txt  tf_wiki75.txt  tf_wiki8.txt
tf_wiki104.txt  tf_wiki18.txt  tf_wiki32.txt  tf_wiki47.txt  tf_wiki61.txt  tf_wiki76.txt  tf_wiki90.txt
tf_wiki105.txt  tf_wiki19.txt  tf_wiki33.txt  tf_wiki48.txt  tf_wiki62.txt  tf_wiki77.txt  tf_wiki91.txt
tf_wiki106.txt  tf_wiki1.txt   tf_wiki34.txt  tf_wiki49.txt  tf_wiki63.txt  tf_wiki78.txt  tf_wiki92.txt
tf_wiki107.txt  tf_wiki20.txt  tf_wiki35.txt  tf_wiki4.txt   tf_wiki64.txt  tf_wiki79.txt  tf_wiki93.txt
```

```
(1048576,[79437,167288,631465,715461,822462,837701],[1.0,1.0,1.0,1.0,1.0,1.0])
```

Below is the results of the output of TF-IDF. If we view the part-00000 file, it shows us the output of TF-IDF. Each number in the file corresponds to the occurrence of words in the Wikipedia articles.

```
bigdata@ubuntu:~/AdvBigData/Homework1/tfIdfVectors$ ls
part-00000  _SUCCESS
```

```
(1048576,[3932,4648,6111,8022,10759,14221,14726,15796,18281,20007,24315,27895,30868,32136,36751,36754,3675
7,43205,49017,50546,50570,50583,52431,57416,58454,59426,66335,77891,79132,82018,83150,87001,87830,92917,93
332,96826,96900,98136,99286,104112,105244,106449,110060,112424,120527,122794,124607,126574,126682,128548,1
30814,131711,133414,138877,141783,142630,143597,144077,144984,146451,147180,148960,162004,169839,177483,17
7621,177703,180063,184591,186579,188382,188693,189356,189688,189690,193916,194428,194853,197442,200642,202
225,211677,212190,212465,216023,216686,220081,227371,228043,229974,229990,230626,231651,232712,238295,2405
44,242311,242620,243071,245793,249907,251806,252578,257324,257979,263739,269616,272037,273047,275410,28114
0,285351,291777,292410,293156,293164,298845,299337,302630,304821,308900,314428,315238,316159,317101,318420
,319398,320458,321542,323297,323305,326588,327088,330975,334322,334392,334904,338770,342344,346681,347424,
349556,351401,353281,355889,356439,360593,362714,367233,372295,374607,375246,376439,377663,382962,382966,3
85284,387119,388382,391549,393139,396916,397848,400266,404461,405332,405374,406040,408376,410494,411410,41
```

3. Use Twitter Streaming API to receive real-time twitter data. Collect 30 mins of Twitter data on 5 companies using keyword=xxx (e.g. ibm). Consider all Twitter data from a company is one document. Create TF-IDF of each company's tweets in 30 minutes.

Next, I used the Twitter Streaming API to get live tweets from each company. I changed the API keys and used the tweepy module to stream Twitter data.

```
from tweepy import Stream
from tweepy import OAuthHandler
from tweepy.streaming import StreamListener
import json
import tweepy
from tweepy.api import API

#company='Facebook'
company = raw_input('Enter company name:')

#consumer key, consumer secret, access token, access secret.
API_KEY="kJoi3GAHdjEIhdfjBqBYt1djd"
API_SECRET="mjIVlABTM08JoEEYgYbrAkch43rraOrLwmawYmhRHxfvoXK191"
ACCESS_TOKEN="989584183-s6utmOQ4S1NAijgPKwlY6nLuE9uE895hssEBmYn8"
ACCESS_TOKEN_SECRET="D01FOFQ5wjswTXtZ5z5gzFFeXx5ghIEzdOWczSad91tEG"

key = tweepy.OAuthHandler(API_KEY, API_SECRET)
key.set_access_token(ACCESS_TOKEN, ACCESS_TOKEN_SECRET)

class Stream2Screen(tweepy.StreamListener):

    def on_status(self, status):
        print status.text.encode('utf8')
        with open(company+' tweets.txt','a') as tf:
                tf.write(status.text.encode('utf8'))
        return True
        self.c = self.c+1
        if self.c < self.n: return True
        else:
            with open(company+' tweets.txt','a') as tf:
                tf.write(str(self.c))
                return True
            print 'tweets = '+str(self.c)
            return False

stream = tweepy.streaming.Stream(key, Stream2Screen())
stream.filter(track=[company])
```

When I ran the above script, I got a live stream of tweets.

```
bigdata@ubuntu:~/AdvBigData/Homework1/Twitter$ python final_twitter.py
Enter company name:Google
SLOW! Test "Cross-Cloud: GET at Google Cloud (IP-address)" response was slow https://t.co/kXC89FFJXV
```

After running the following script for 30 minutes, it generates a tweets text file. This is the output of the tweets file for Google.

```
you know you are desperate when you go to the second page of GoogleRT @VIralBuzzNewss: Google Translate No
w Has A lot more Than 100 Languages And Covers 99 Percent Of The On... -  https://t.co/7ncse5oFP1 ht…RT @T
echCrunch: Google's CEO Says "Forcing Companies To Enable Hacking Could Compromise Users' Privacy." https:
//t.co/XddKIDpfXk by @mjburn…RT @karaswisher: Google CEO @sundarpichai says FBI request for Apple back doo
r sets "troubling precedent" https://t.co/H0OhpN1EXg via mhber…RT @JeremyCorbyn4PM: Not just #matesrates f
or Google but a freebie for family. George Osborne continuing to help people avoid tax. https://…RT @iMac_
```

Then, I conducted TF and TF-IDF on the company. For Macy's for example, we get the following results for TF:

```
(1048576,[5438,6292,8148,12928,16682,19527,21465,22438,22546,24081,24165,26942,30117,33950,34574,36748,36
51,36757,38019,38034,44447,44941,45640,50570,50573,50583,53070,53208,55144,57166,58115,62184,66906,75344,
5612,78427,86525,90996,94984,101357,108104,110444,110522,111879,112640,114629,123946,124475,125122,125479
131792,133414,134398,138807,147473,166148,166513,167368,171345,173019,175708,175822,175866,180059,180990,
84466,185704,187352,187458,189356,192424,193475,196531,197814,207249,215856,216290,220729,226483,230074,2
0431,232338,234013,240455,241438,242958,244514,245347,246947,248433,249974,253320,253484,254072,261936,26
724,263739,264499,267578,276347,282612,287749,290859,293156,295006,296279,302646,315457,315618,319194,319
30,323297,323305,326993,328912,329514,334395,337549,343316,343660,347143,360881,364111,372544,373461,3860
8,387133,387765,389896,392560,396003,397392,398564,401499,409190,409491,420843,422488,423407,428883,43094
,434754,434774,438051,442679,443863,445774,447348,449293,452519,457035,464019,469882,470450,472360,474906
477121,477623,485635,485808,485821,492610,497246,510753,511574,512198,512935,514653,516742,520772,524007,
24835,525238,525630,529891,530678,531940,535338,535609,551305,551915,558681,559923,565064,566674,574099,5
7177,578538,581984,582329,586589,589643,591068,595821,595827,597939,600868,603449,603455,605454,605619,60
964,608735,614318,617741,629096,631833,634034,634970,641924,645283,648260,652355,659368,661664,669585,673
88,678547,690466,692394,697419,699067,701180,702676,706954,709976,714910,715648,718525,719560,725041,7262
0,738826,749730,751615,756653,757550,784665,786878,791942,795365,796752,800297,801405,801557,809186,80987
,815906,821963,824454,828458,829925,832474,839292,847183,848594,854516,856697,863562,880002,882630,883976
886775,894772,897504,910182,914068,914281,914606,916424,920079,922684,925721,928288,931063,944044,945600,
51974,959994,961085,972429,974198,974703,976148,977868,982000,986393,990296,991410,991597,993084,1000874,
004171,1005458,1007180,1008456,1017476,1020163,1022085,1025069,1029305,1030127,1033734,1034508,1034670,10
7677,1037854,1043959,1046706,1047513],[7.0,6.0,1.0,4.0,1.0,1.0,1.0,1.0,3.0,1.0,6.0,1.0,1.0,1.0,1.0,1.0,1.
,12.0,2.0,2.0,1.0,1.0,1.0,2.0,4.0,7.0,1.0,1.0,1.0,3.0,1.0,4.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,4.0
2.0,1.0,1.0,3.0,1.0,1.0,2.0,1.0,3.0,1.0,1.0,1.0,2.0,1.0,14.0,1.0,2.0,1.0,1.0,1.0,1.0,1.0,3.0,1.0,1.0,2.0,
.0,1.0,6.0,1.0,1.0,1.0,1.0,2.0,2.0,2.0,2.0,1.0,1.0,2.0,1.0,4.0,1.0,6.0,4.0,1.0,1.0,1.0,4.0,1.0,5.0,3.0,4.
,1.0,1.0,1.0,1.0,2.0,4.0,1.0,3.0,1.0,1.0,1.0,1.0,1.0,2.0,1.0,3.0,4.0,4.0,1.0,1.0,1.0,3.0,1.0,1.0,2.0,1.0,
.0,1.0,1.0,4.0,2.0,1.0,2.0,1.0,1.0,4.0,1.0,2.0,3.0,1.0,2.0,1.0,1.0,2.0,2.0,1.0,3.0,1.0,1.0,1.0,1.0,1.0,1.
,1.0,2.0,1.0,1.0,2.0,1.0,1.0,2.0,1.0,1.0,1.0,1.0,1.0,2.0,1.0,1.0,7.0,1.0,3.0,4.0,1.0,2.0,1.0,1.0,1.0,6.0,
.0,2.0,1.0,1.0,1.0,1.0,1.0,2.0,1.0,2.0,1.0,3.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,4.0
,2.0,1.0,2.0,1.0,2.0,3.0,1.0,2.0,1.0,1.0,2.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,2.0,3.0,1.0,2.0,1.0,6.0,10.0
1.0,2.0,11.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,27.0,1.0,1.0,1.0,1.0,1.0,4.0,1.0,1.0,1.0,1.0,1.0,4.0,3.0,1.0,2.0,1.0
1.0,1.0,2.0,2.0,1.0,2.0,1.0,2.0,1.0,1.0,24.0,1.0,2.0,1.0,4.0,1.0,3.0,1.0,1.0,1.0,1.0,1.0,1.0,7.0,7.0,2.0,
.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,5.0,2.0,1.0,1.0,1.0,2.0,1.0,2.0,2.0,1.0,1.0,2.0,1.0,1.0,1.0,2.0,2.0
,1.0,1.0,1.0])
```

Additionally, for Macy's, we get the following for TF-IDF:

```
(1048576,[5438,6292,8148,12928,16682,19527,21465,22438,22546,24081,24165,26942,30117,33950,34574,36748,36
51,36757,38019,38034,44447,44941,45640,50570,50573,50583,53070,53208,55144,57166,58115,62184,66906,75344,
5612,78427,86525,90996,94984,101357,108104,110444,110522,111879,112640,114629,123946,124475,125122,125479
131792,133414,134398,138807,147473,166148,166513,167368,171345,173019,175708,175822,175866,180059,180990,
84466,185704,187352,187458,189356,192424,193475,196531,197814,207249,215856,216290,220729,226483,230074,23
0431,232338,234013,240455,241438,242958,244514,245347,246947,248433,249974,253320,253484,254072,261936,263
724,263739,264499,267578,276347,282612,287749,290859,293156,295006,296279,302646,315457,315618,319194,319
30,323297,323305,326993,328912,329514,334395,337549,343316,343660,347143,360881,364111,372544,373461,3860
8,387133,387765,389896,392560,396003,397392,398564,401499,409190,409491,420843,422488,423407,428883,43094
,434754,434774,438051,442679,443863,445774,447348,449293,452519,457035,464019,469882,470450,472360,474906
477121,477623,485635,485808,485821,492610,497246,510753,511574,512198,512935,514653,516742,520772,524007,
24835,525238,525630,529891,530678,531940,535338,535609,551305,551915,558681,559923,565064,566674,574099,57
7177,578538,581984,582329,586589,589643,591068,595821,595827,597939,600868,603449,603455,605454,605619,607
964,608735,614318,617741,629096,631833,634034,634970,641924,645283,648260,652355,659368,661664,669585,6732
88,678547,690466,692394,697419,699067,701180,702676,706954,709976,714910,715648,718525,719560,725041,7262
0,738826,749730,751615,756653,757550,784665,786878,791942,795365,796752,800297,801405,801557,809186,80987
,815906,821963,824454,828458,829925,832474,839292,847183,848594,854516,856697,863562,880002,882630,883976
886775,894772,897504,910182,914068,914281,914606,916424,920079,922684,925721,928288,931063,944044,945600,
51974,959994,961085,972429,974198,974703,976148,977868,982000,986393,990296,991410,991597,993084,1000874,
004171,1005458,1007180,1008456,1017476,1020163,1022085,1025069,1029305,1030127,1033734,1034508,1034670,103
7677,1037854,1043959,1046706,1047513],[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0
.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0
.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0
.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0
.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0
.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0
.0,0.0])
```


I generated the data for the following companies: Facebook, General Dynamics, Google, IBM and Macy's as shown below.

```
bigdata@ubuntu:~/AdvBigData/Homework1/Twitter$ ls
Facebook tweets.txt           tf_General Dynamics tweets1.txt  tfIdfMacys tweets
final_twitter.py              tf_Google tweets1.txt            tfidf_twitter.py
General Dynamics tweets.txt   tf_IBM tweets1.txt               tfIdfVectors
Google tweets.txt             tfIdfFacebook tweets             tf_Macys tweets1.txt
IBM tweets.txt                tfIdfGeneral Dynamics tweets     Twitter tweets.txt
Macys tweets.txt              tfIdfGoogle tweets
tf_Facebook tweets1.txt       tfIdfIBM tweets
```

As shown, Facebook had the most amount of tweets in 30 minutes and General dynamics had the least amount of tweets.

```
-rw-rw-r-- 1 bigdata bigdata 2048884 Feb 17 19:27 Facebook tweets.txt
-rw-rw-r-- 1 bigdata bigdata     178 Feb 17 19:20 General Dynamics tweets.txt
-rw-rw-r-- 1 bigdata bigdata 1273214 Feb 18 15:13 Google tweets.txt
-rw-rw-r-- 1 bigdata bigdata   52003 Feb 17 18:54 IBM tweets.txt
-rw-rw-r-- 1 bigdata bigdata    4063 Feb 17 20:13 Macys tweets.txt
```

4. Use Yahoo Finance to receive the Stock price data. Collect 30 mins of Finance data on 5 companies, one value per minute. Use the outlier function to display outliers that are larger than two standard deviation.

Lastly, I used the Yahoo! Finance module to get the stock price data for 5 different companies.
I created a script to get the Yahoo Finance data for 30 minutes.

```
import time
import json
from yahoo_finance import Share

share = raw_input('Enter company abbreviation: ')
stocks_price = []

def getPrice(share):
        for x in range(0,30):
                price = Share(share)
                f = open(share+".txt",'a')
                        f.write(price.get_price()+'\n')
                f.close()
                stocks_price.append(price.get_price)
                time.sleep(60)
        print 'Share info recieved'

result = getPrice(share)
```

```
bigdata@ubuntu:~/AdvBigData/Homework1/Finance$ python yahoofinance.py
Enter company abbreviation: YHOO
```

The script generates a text file that stores the stock price every minute for 30 minutes. These are some of the results I got for Yahoo!:

```
29.73
29.69
29.7399
29.73
29.71
29.73
29.7399
29.785
29.7799
29.7899
29.81
29.76
29.75
29.7199
```

I found the stock prices for IBM, Lockheed Martin (LMT), Yahoo (YHOO), Bank of America (BAC) and Apple (APPL).

After finding the stock prices, I found the outliers in the data. I created an outlier script as shown below:

```
import math
from pyspark import SparkContext

company = "BAC"
sc = SparkContext()
def arg(w):
        try:
                return float(w[0])
        except (ValueError,TypeError):
                return 0
txtfile = sc.textFile("/home/bigdata/AdvBigData/Homework1/Finance/"+company+".tx
t").map(lambda line: line.split(' '))
distance = txtfile.map(arg)
stats = distance.stats()
mean = stats.mean()
stdev = stats.stdev()
outliers = distance.filter(lambda x: math.fabs(x - mean) > 2 * stdev)
print outliers.collect()
```

After entering the company name in the script above, I ran the script on Spark as shown below. I outputted the results into a text file.

```
bigdata@ubuntu:~/AdvBigData/Homework1/Finance$ /usr/local/spark-1.6.0-bin-hadoop
2.4/bin/spark-submit /home/bigdata/AdvBigData/Homework1/Finance/final_outliers.p
y > BAC_outliers.txt
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
16/02/19 13:44:22 INFO SparkContext: Running Spark version 1.6.0
16/02/19 13:44:22 WARN NativeCodeLoader: Unable to load native-hadoop library fo
r your platform... using builtin-java classes where applicable
```

There were no outliers for Bank of America. The output displayed an empty array. This makes sense because the data ranges consistently from 12.36 to 12.46. When I selected another company, such as Lockheed Martin (LMT), I got the following outliers:

```
[214.25, 214.33]
```

The dataset for Lockheed Martin looks like the following:

```
215.06
214.993
215.30
215.4312
215.257
215.34
215.1714
215.14
214.97
214.92
214.78
214.79
214.84
215.08
215.18
215.15
215.29
215.07
215.16
215.08
214.891
214.751
214.75
214.93
214.73
214.78
214.4726
214.25
214.33
214.58
```

As you can see, the outliers are the two lowest values in the dataset, and are larger than two standard deviation.