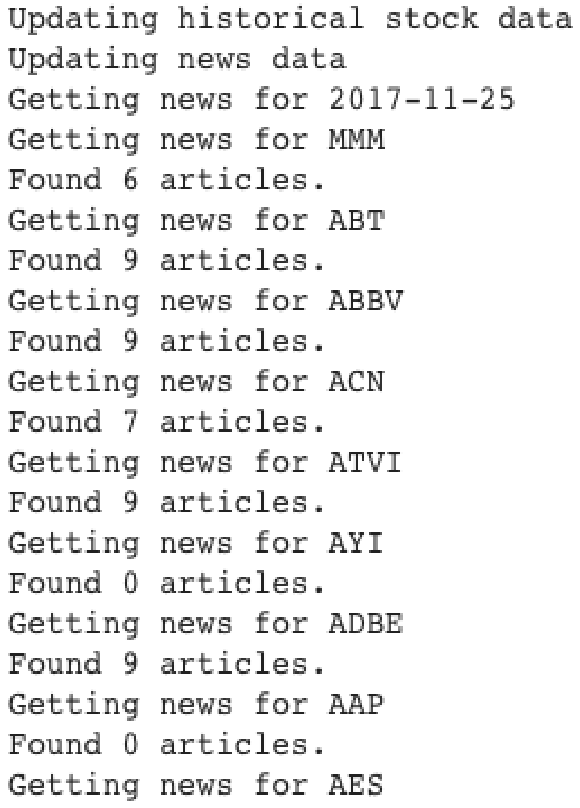
Stock Price Prediction Using Text Analysis

Ankit Panda, Abhinandan Bhatnagar, Aman Mittal, Nikhil Mahipal
Rahul Garg, Rishabh Bansal

Department of Computer Science and Engineering
Bennett University, Greater Naoida, UP

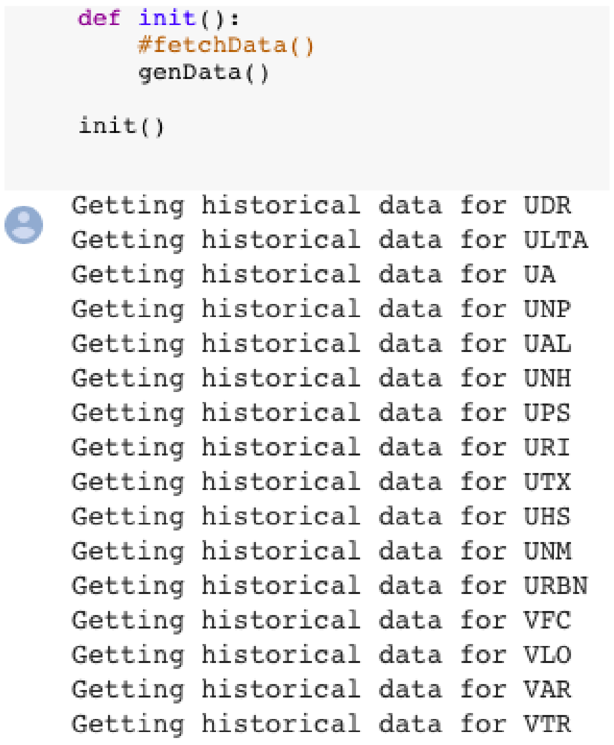
1 Experimental Results

For the Stock Prediction we have used google's Colaboratory, it is a jupyter notebook environment that runs entirely in the cloud, this aids our systems in heavy processing power. The data that we fetch are of two kinds: The Numerical data and the corresponding news articles for the particular stock. The following shows the fetching of the articles.



```
Updating historical stock data
Updating news data
Getting news for 2017-11-25
Getting news for MMM
Found 6 articles.
Getting news for ABT
Found 9 articles.
Getting news for ABBV
Found 9 articles.
Getting news for ACN
Found 7 articles.
Getting news for ATVI
Found 9 articles.
Getting news for AYI
Found 0 articles.
Getting news for ADBE
Found 9 articles.
Getting news for AAP
Found 0 articles.
Getting news for AES
```

The following piece of code fetches the data for us i.e. the Historical data for the stocks.

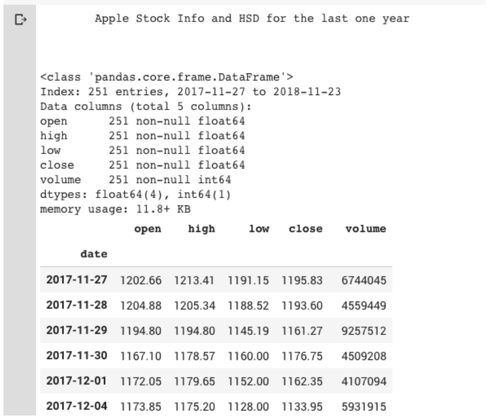


```
def init():
    #fetchData()
    genData()

init()
```

```
Getting historical data for UDR
Getting historical data for ULTA
Getting historical data for UA
Getting historical data for UNP
Getting historical data for UAL
Getting historical data for UNH
Getting historical data for UPS
Getting historical data for URI
Getting historical data for UTX
Getting historical data for UHS
Getting historical data for UNM
Getting historical data for URBN
Getting historical data for VFC
Getting historical data for VLO
Getting historical data for VAR
Getting historical data for VTR
```

The Historical data and the articles are then stored date wise for a total of 1 year ranging from 27th Nov 2017 to 23 Nov 2018. The Following is an example of the APPL stock info for the past one year. The dataframe contains: open, high, low, close volume.

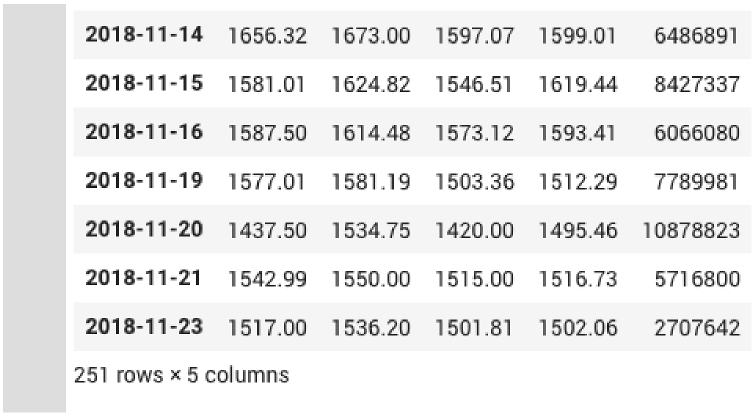


```
        Apple Stock Info and HSD for the last one year



<class 'pandas.core.frame.DataFrame'>
Index: 251 entries, 2017-11-27 to 2018-11-23
Data columns (total 5 columns):
open      251 non-null float64
high      251 non-null float64
low       251 non-null float64
close     251 non-null float64
volume    251 non-null int64
dtypes: float64(4), int64(1)
memory usage: 11.8+ KB
```

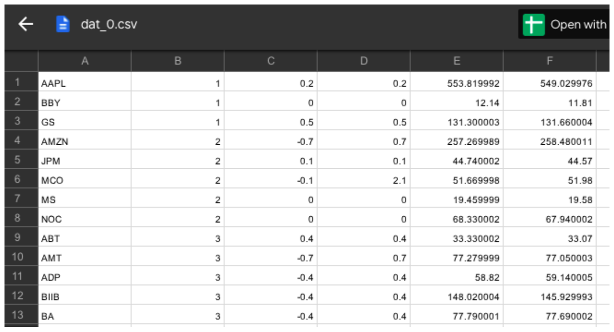
	open	high	low	close	volume
date					
2017-11-27	1202.66	1213.41	1191.15	1195.83	6744045
2017-11-28	1204.88	1205.34	1188.52	1193.60	4559449
2017-11-29	1194.80	1194.80	1145.19	1161.27	9257512
2017-11-30	1167.10	1178.57	1160.00	1176.75	4509208
2017-12-01	1172.05	1179.65	1152.00	1162.35	4107094
2017-12-04	1173.85	1175.20	1128.00	1133.95	5931915



2018-11-14	1656.32	1673.00	1597.07	1599.01	6486891
2018-11-15	1581.01	1624.82	1546.51	1619.44	8427337
2018-11-16	1587.50	1614.48	1573.12	1593.41	6066080
2018-11-19	1577.01	1581.19	1503.36	1512.29	7789981
2018-11-20	1437.50	1534.75	1420.00	1495.46	10878823
2018-11-21	1542.99	1550.00	1515.00	1516.73	5716800
2018-11-23	1517.00	1536.20	1501.81	1502.06	2707642

251 rows × 5 columns

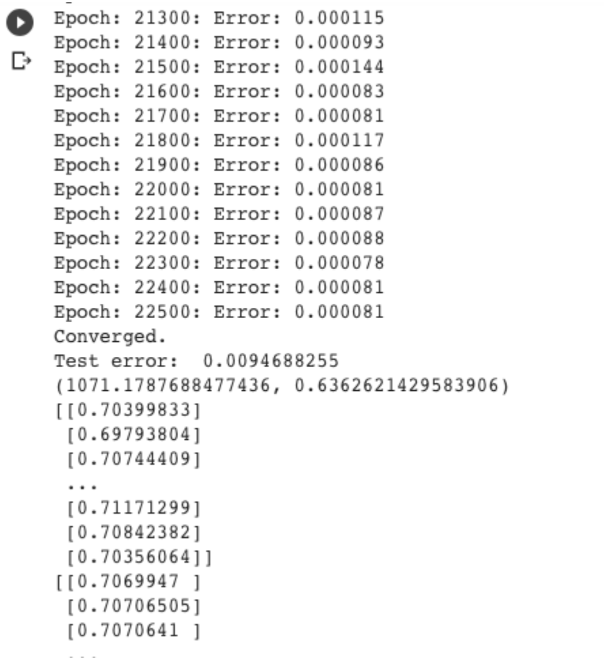
After Collecting all the prerequisite data for processing we use the Function Gendata to produce the Sentiment analysis of the article, Subjectivity of the Article, opening price of the Stock and finally the opening price of the stock after the article was published.



dat_0.csv

	A	B	C	D	E	F
1	AAPL	1	0.2	0.2	553.819992	549.029976
2	BBY	1	0	0	12.14	11.81
3	GS	1	0.5	0.5	131.300003	131.660004
4	AMZN	2	-0.7	0.7	257.269989	258.480011
5	JPM	2	0.1	0.1	44.740002	44.57
6	MCO	2	-0.1	2.1	51.669998	51.98
7	MS	2	0	0	19.459999	19.58
8	NOC	2	0	0	68.330002	67.940002
9	ABT	3	0.4	0.4	33.330002	33.07
10	AMT	3	-0.7	0.7	77.279999	77.050003
11	ADP	3	-0.4	0.4	58.82	59.140005
12	BIIB	3	-0.4	0.4	148.020004	145.929993
13	BA	3	-0.4	0.4	77.790001	77.690002

The Data Produced above is used in our neural network which breaks the data into train and test data to use it the our Google's Tensorflow Model. After Training the model for about 22,000 epochs the test error converges at 0.0094688255

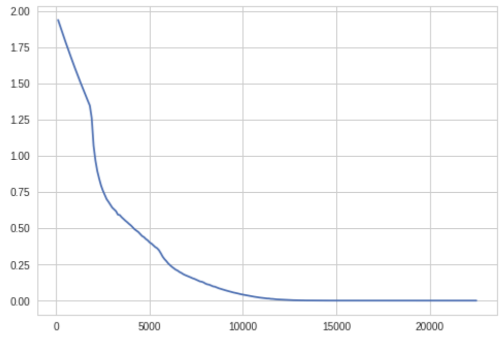


```
Epoch: 21300: Error: 0.000115
Epoch: 21400: Error: 0.000093
Epoch: 21500: Error: 0.000144
Epoch: 21600: Error: 0.000083
Epoch: 21700: Error: 0.000081
Epoch: 21800: Error: 0.000117
Epoch: 21900: Error: 0.000086
Epoch: 22000: Error: 0.000081
Epoch: 22100: Error: 0.000087
Epoch: 22200: Error: 0.000088
Epoch: 22300: Error: 0.000078
Epoch: 22400: Error: 0.000081
Epoch: 22500: Error: 0.000081
Converged.
Test error:  0.0094688255
(1071.1787688477436, 0.6362621429583906)
[[0.70399833]
 [0.69793804]
 [0.70744409]
 ...
 [0.71171299]
 [0.70842382]
 [0.70356064]]
[[0.7069947 ]
 [0.70706505]
 [0.7070641 ]
 ...
```

The Following is the plot for the error and it's corresponding epochs. Here We observe that while the initial Epochs held the error of about 2 we have reached the saturation point at somewhere at 12000 epochs where it has come down to about 0.009.

```
[11] y= np.arange(100, len(error)*100+1, 100)
     plt.plot(y,error)
```

```
[<matplotlib.lines.Line2D at 0x7fc4173bfb10>]
```



2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
5000
10000
15000
20000