Sentiment Analysis of Twitter to Predict Stock Market

Yuechen Jiang - October 30, 2020

Reference

https://arxiv.org/pdf/1010.3003.pdf

https://link.springer.com/article/10.1057/s41265-016-0034-2

http://blueanalysis.com/iulianserban/Files/twitter_report.pdf

http://cs229.stanford.edu/proj2011/GoelMittalStockMarketPredictionUsingTwitterSentimentAnalysis.pdf

https://github.com/cjhutto/vaderSentiment

https://pbpython.com/excel-file-combine.html

https://www.data-blogger.com/2017/02/24/gathering-tweets-with-python/

https://stackoverflow.com/questions/44948628/how-to-take-all-tweets-in-a-hashtag-with-tweepy

Introduction

The Data Cleaning part and the SVM model use the codes of other projects. This project adds KNN, Logistics Regression and Naive Bayes models on the basis of previous projects, and improves the backtest.

Positive and negative emotions in social media messages, such as Twitter, can be used to predict daily changes or trends in stock prices.

Although news definitely affects stock market prices, public sentiment may also play an equally important role. We know from psychological research that emotions, like information, play an important role in human decision-making. Behavioral finance further proves that financial decisions are largely driven by emotions. Therefore, we have reason to assume that public sentiment can drive stock market prices like news.

Today's Tweet carries positive or negative sentiment, and contains one or several cashtags that can affect the trend of the stock tomorrow. If negative sentiment prevails today, then stock prices are expected to fall tomorrow, and vice versa. The number of followers on the Twitter account is also a major factor. The more followers an account has, the greater the influence of tweets, and the greater the impact of their emotions on stock prices.

Data Collecting

Scrape Tweets from Twitter using Python and Tweepy

Tweets are extremely useful for gathering opinions of thousands of people on a particular topic over time. Sadly, Twitter has revoked access to old Tweets (however, this Python package is still capable of doing so by making use of Twitter search functionality). Therefore, many developers harvest Tweets by using Twitters Streaming API and store them on their computing nodes. If you have enough computing nodes, you could consider collecting Tweets by using a cluster and cluster software, such as Apache Spark or Apache Flink. But if you have a small scale project, one Python script will be enough. In this tutorial, we will build a small Python script for retrieving and storing Tweets from the Streaming API.

Setting up an account

The first thing we need, is an access token for accessing the Twitter API. This can simply be done by visiting apps.twitter.com. Sadly, the policy and terms of service of Twitter changes frequently, so it is hard to explain and update the sign up process on Twitter every time Twitter changes something.

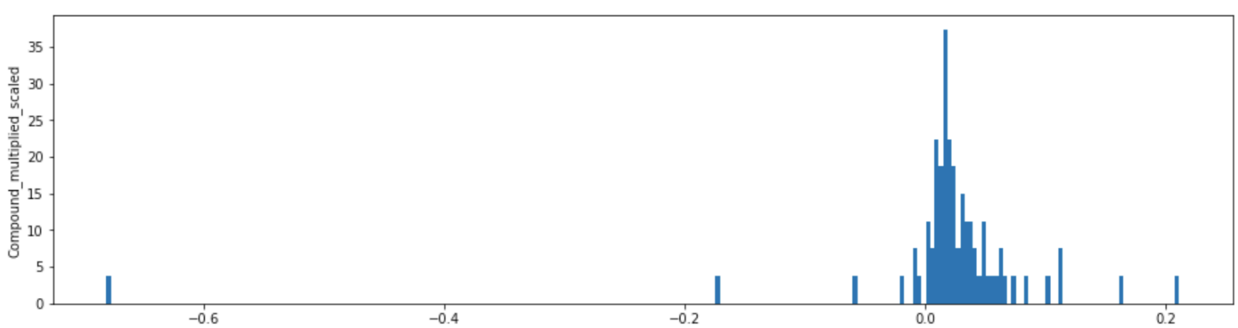
Data Choosing

From March 28, 2019 to June 15, 2019, about 1 million tweets were collected in 79 days, which mentioned the cashtags of the companies in the Nasdaq 100 index. In order to extract the sentiment of each tweets, we use VADER.

After combining each tweet with its sentiment, multiply it by the number of followers for that account. In this way, in the final model, the sentiment of more “influential” accounts will be given more weight. After that, these tweets (an average of 6,500 per cashtags) were compressed into 75 lines, which included the daily average of each sentiment, and then compared with the daily price changes of the relevant stocks.

Use Python's pandas-datareader library to download daily stock data from Yahoo Finance. After adding the daily percentage change column to the stock data and interpolating the missing data over the weekend, it is now possible to merge the two data sets, namely the sentiment of the tweet and the daily change of the stock.

Compound multiplied scaled -features distributed



Flow Description

Stock data

1. Download historical stock price data from Yahoo Finance Time period March-June 2019;

2. Add a new column of daily percentage;

changes of the stock prices; Oct_change_stock;

3. Interpolate stock data over weekends to make it compatible with the daily occurring tweet data.

Twitter data

1. Download the tweets for each cashtag, time period March-June 2019;

2. Run the text in each tweet through the sentiment analyzer using the Vader library in Python;

3. Combine the sentiments extracted with Vader to the tweets;

4. Clean the data; remove unnecessary features, align dates throughout the timeseries, remove tweets where sentiment is neutral, replace missing data etc.

5. Creat a new feature, Compound_multiplied, where tweet sentiment is multiplied by the number of followers of the account. Logic being, that the more followers an account has, the bigger impact on the stocks should also the sentiment of its tweets have;

6. Scale this new feature Compounf_multiplied to prepare it for the machine learning algorithms;

7. Create a new dataframe of daily averages of the Compound_multiplied feature;

8. Create a new dataframe with Compound_multiplied and Pec_change_stock;

9. Creat a new column Buy/Sell to the data frame, which is [-1] if stock went down on a day and [+1] if stock went up. Shoft each value one step up to the previous day. This is what the final model will predict.

Machine learning classifier

Since this is a binary classification task, that is, the result is either "buy" or "sell", so we use 4 such algorithms:

- KNN K-Nearest-Neighbors(KNN)
- Logistic regression(LogReg)
- Support Vector Machine (SVM)
- Naive Bayes

Training/test data split

Among the 74 days of available data, 59 days (80%) of data for each stock are used for training, and 15 days (20%) of data are used to test the accuracy of each algorithm.

Cross-validation

Due to the limited amount of data, using only 20% of the data (15 days) and 80% of the training data (59 days) for testing may not be representative. In order to avoid the possibility that the training/test split is not completely random, the data is cross-validated, so that a more representative result of the accuracy of each algorithm is obtained. The training data is further divided into 10 subsets, and each subset is tested against the other 9 subsets.

1. Make a Train/Test-split(80/20)of the time-series data covering 74 days; Data from 59 days (80%) will be used for training and 15 days (20%) will be used for testing.

- Features: Compound_multiplied and Pct_change_stock
- Label: Buy/Sell

2. Train the four classifier algorithms with the data for each cashtag:

- KNN K-Nearest-Neighbors(KNN)
- Logistic regression(LogReg)
- Support Vector Machine (SVM)
- Check the accuracy of each algorithm for the data with 10-fold cross validation.

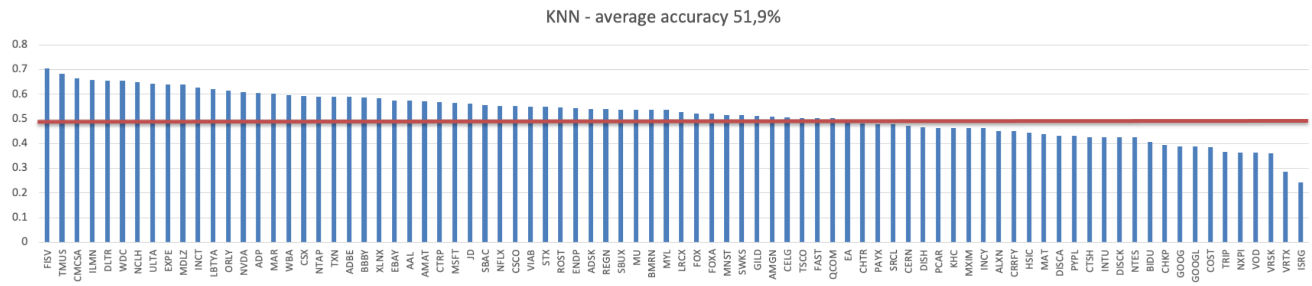
3. Collect the results and make a summarized overview; which models and cashtags performed best.

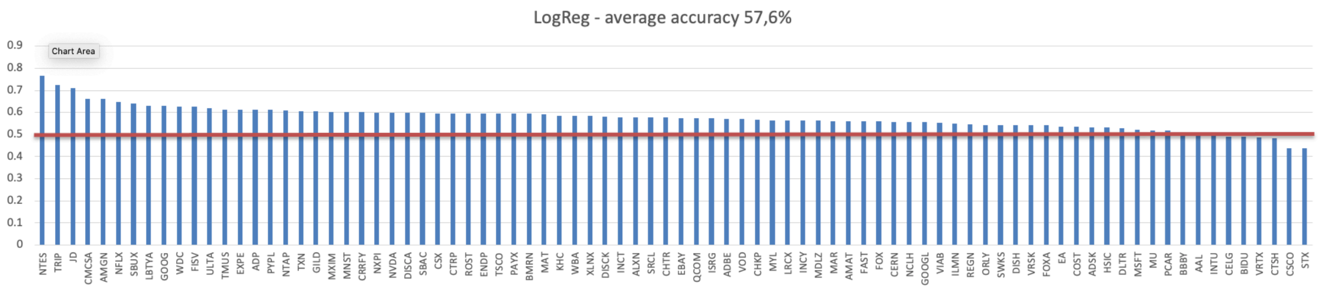
Model results

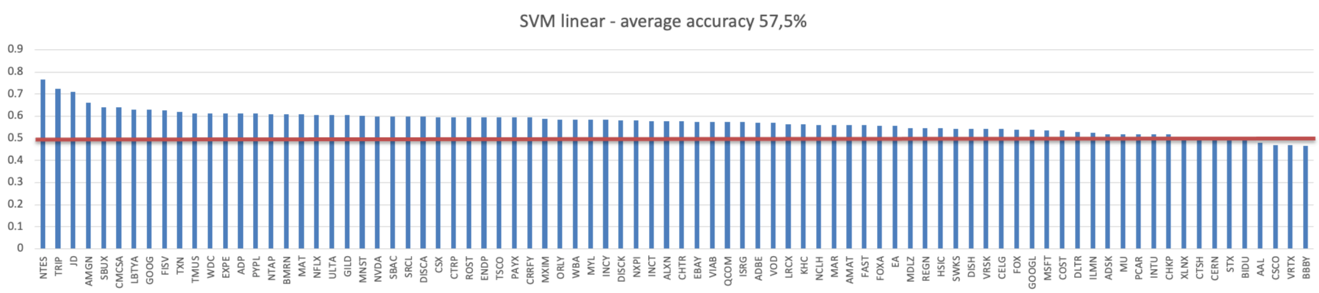
After passing the stock through 4 binary classifiers and performing 10-fold cross-validation, the results are as follows. The average accuracy of each classifier is higher than 50%. This means that sentiment on Twitter is predictable, at least better than coin flips. The average accuracy of coin tossing is 50%, so the accuracy of more than 50% proves to a certain extent the ability of the model to obtain "extraordinary" returns.

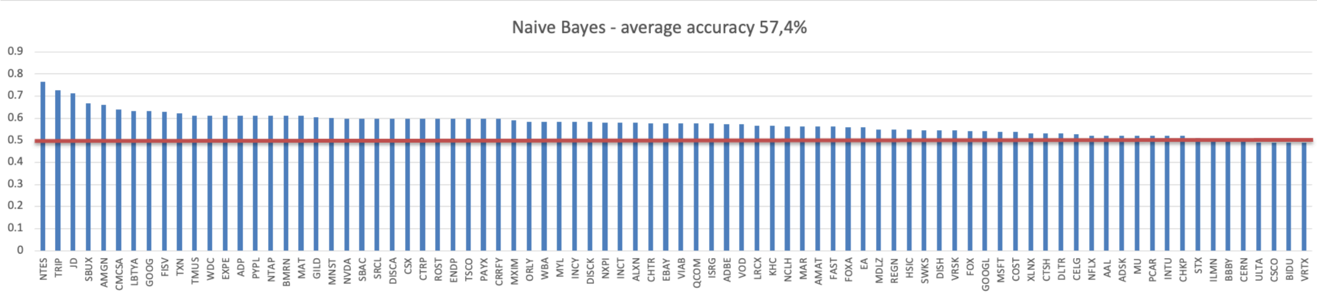
Accuracy after cross validation

KNN	0.575238095238
LogReg	0.627619047619
SVM Linear	0.627619047619
Naive Bayes	0.627619047619


KNN - average accuracy 51,9%
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0


LogReg - average accuracy 57,6%
Chart Area
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0


SVM linear - average accuracy 57,5%
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0


Naive Bayes - average accuracy 57,4%
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0

Back-testing

Next, we compared the profit and loss of the simple buy and hold strategy with the profit and loss achieved using the model. We selected 8 NASDAQ stocks for simulation, and the tweet data of the simulated trading in March was collected by "crawling" Twitter using its Developer API. We downloaded all tweets containing cashtags $AAL, $ADP, $CERN, $EXPE, $FISV, $TMUS, $TXN and $WDC in March 2020.

1. Tweets run through sentiment analysis algorithms, and each tweet has a sentiment; positive, neutral or negative.

2. Each tweet is multiplied by the number of followers of the account. In this way, in the final model, the sentiment of the more "influential" accounts will get more weight.

3. Tweet data is compressed into 28 rows, including the daily average of each sentiment, and compared with the daily price changes of related stocks during the same period.

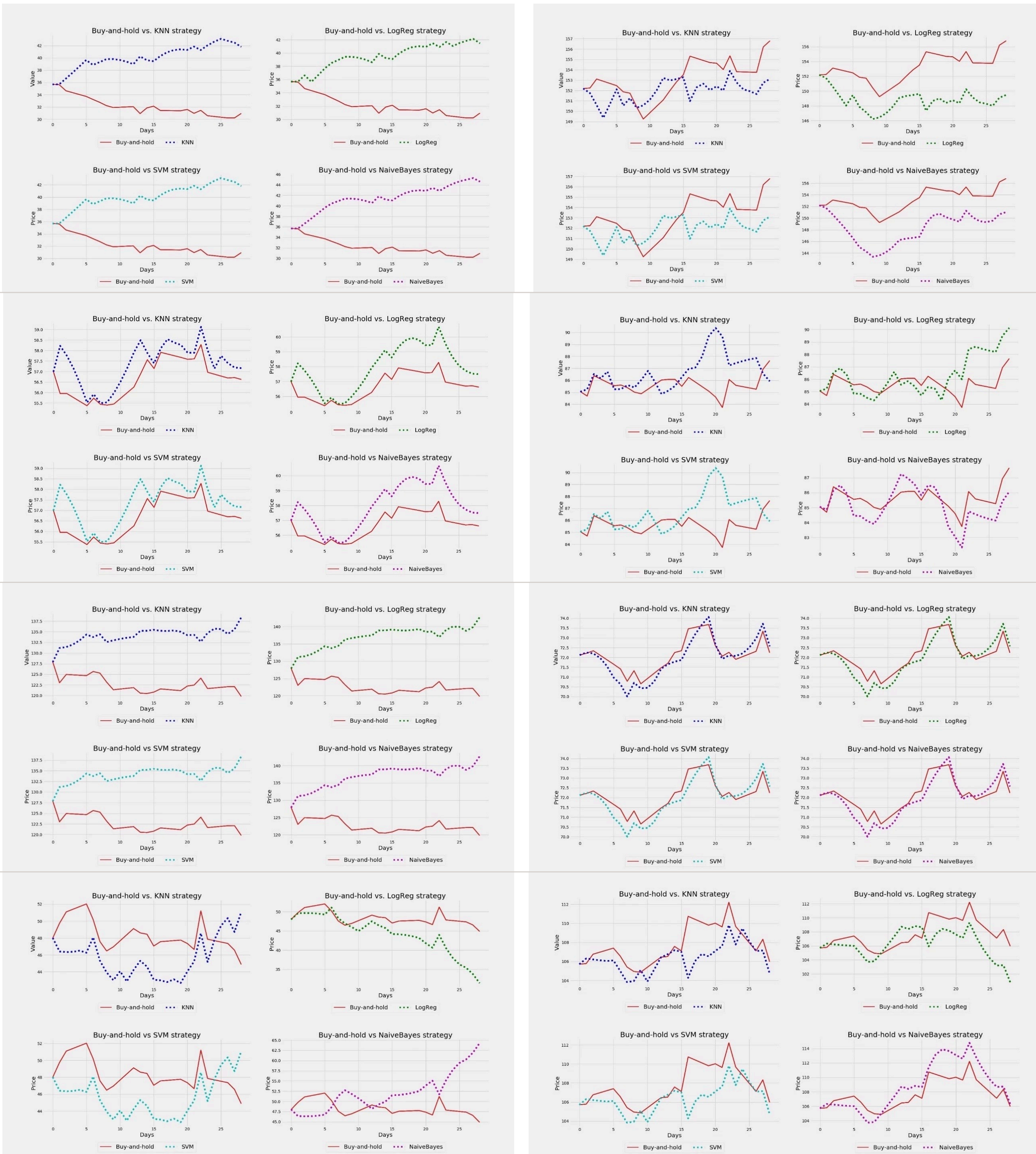
4. Download the stock data and add it to the "Daily Change Percentage" column.

5. Combine Tweet and stock data, and add a label column, namely "buy or sell". This is what the model is trying to predict.

These data sets are then used by comparing the buy-and-hold strategy with four different models. Each daily expected daily stock price change is predicted using the model.

Conduct simulated trading March 2020

A buy and hold strategy was adopted for 8 stocks, and compared with the other 4 strategies based on binary classification algorithm.



Further improve the model ideas

1. The model only has 75 days of data for training and testing. If emotions are truly predictive, adding more data from a longer or even more recent period may significantly improve the results.

2. Consider using the results of Twitter sentiment in combination with other technologies, such as LSTM neural network for time series analysis, and always make predictions one day in advance.

3. Try to use some other ready-made models, such as TextBlob, instead of VADER to extract tweet sentiment. Or a better way is to train the emotion classifier by building a neural network and then use the data to train it.

Appendix:

The code used for the result:

```
# cashtag_scraper

import tweepy
import pandas as pd
import xlsxwriter
import os

consumer_key= ["hQGFxHs1WgKUjY2oSglp6pLOE"]
consumer_secret= ["47L4hAVj0VgYrtqNIHf4mOjggZnXfgmNlABGjCDarqJpEqgx6Q"]
access_key=
["AAAAAAAAAAAAAAAAAAAAAF8aJQEAAAAAQ4gfWtBV8QFu4TEz7hHL%2FU3fDsY%3
DyhqiLLeTWTt9kZ5vm2jtI1qpJSQS01LpvkXS0Pmt9cMUMyF0Is"]
access_secret= ["47L4hAVj0VgYrtqNIHf4mOjggZnXfgmNlABGjCDarqJpEqgx6Q"]

#Twitter Access
auth = tweepy.OAuthHandler( consumer_key,consumer_secret)
auth.set_access_token(access_key,access_secret)
api = tweepy.API(auth,wait_on_rate_limit = True

df = pd.DataFrame()
msgs = []
msg =[]

for tweet in tweepy.Cursor(api.search, q='$AAL', rpp=100, tweet_mode="extended").items(3000):
    #msg = [tweet.created_at, tweet.text, tweet.retweet_count, tweet.favorite_count]
    msg = [tweet.created_at, tweet.full_text, tweet.user.followers_count] #choose the twitter data for
your df
    msg = tuple(msg)
    msgs.append(msg)

df = pd.DataFrame(msgs)

df.columns = ['created at', 'text', 'follower count']
df.tail()

writer_df = pd.ExcelWriter('df.xlsx', engine='xlsxwriter')
df.to_excel(writer_df)
writer_df.save()

os.rename('df.xlsx', '$AAL_tweets.xlsx') # Update cashtag!
```

```
# Data cleaning and feature engineer

%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
import datetime as dt
from sklearn.preprocessing import StandardScaler
import pandas_datareader.data as web
import math
import xlsxwriter
import os

xls = pd.ExcelFile('/Users/yuechenjiang/Desktop/FE690/Tweets/AAL.xlsx') # CHANGE FILE
NAME!!!

stock = "AAL" #CHANGE STOCK TICKER!!!

df = pd.read_excel(xls, header = 0,encoding='latin-1', sheet_name = "Stream")

df.head()

Tweet = df['Tweet content']
Tweet.head()

def sentimentScore(Tweet):
    analyzer = SentimentIntensityAnalyzer()
    results = []
    for sentence in Tweet:
        vs = analyzer.polarity_scores(sentence)
        print("Vader score: " + str(vs))
        results.append(vs)
    return results

df_results = pd.DataFrame(sentimentScore(Tweet))

df_tweets = pd.merge(df, df_results, left_index=True, right_index=True)
```

```
df_tweets.tail()

df_tweets = df_tweets[(df_tweets['Date'] >= '2016-04-01') & (df_tweets['Date'] <= '2016-06-14')]

df_tweets.tail()

df_tweets['datetime'] = pd.to_datetime(df_tweets['Date']) # change of Date column to datetime
columns
df_tweet_SA = df_tweets.set_index('datetime') # creates a new dataframe 'df_Dc' with the new index
column datetime
df_tweet_SA.drop(['Date'], axis=1, inplace=True) #drops the original 'Date' column from the dataframe
df_tweet_SA.head()

df_tweet_SA = df_tweets[['Date','Hour','Tweet content','Favs','RTs','Followers','Following', 'Is a RT',
            'Hashtags','Symbols','compound','neg','neu','pos','datetime']]

df_tweet_SA.head()

df_tweet_SA = df_tweet_SA[(df_tweet_SA[['compound']] != 0).all(axis=1)]

df_tweet_SA['Compound_multiplied'] = df_tweet_SA['compound']*df_tweet_SA['Followers']

df_tweet_SA.tail()

nan_rows = df_tweet_SA[df_tweet_SA['Followers'].isnull()]
nan_rows

df_tweet_SA = df_tweet_SA[np.isfinite(df_tweet_SA['Followers'])]

from sklearn.preprocessing import StandardScaler

x_1 = df_tweet_SA[['Compound_multiplied']].values.astype(float)

scaler = StandardScaler().fit(x_1)

scaled_data = scaler.transform(x_1)

df_tweet_SA['Compound_multiplied_scaled'] = scaled_data

df_tweet_SA.tail()
len(df_tweet_SA)
```

```
df_daily_mean=(df_tweet_SA.groupby(df_tweet_SA.datetime).mean())

df_daily_mean.tail()

len(df_daily_mean)

# Download stock data from yahoo finance

import pandas_datareader.data as web

start = dt.datetime(2019, 4, 2)
end =  dt.datetime(2019, 6, 14) #dt.datetime.now()

df_stock = web.DataReader(stock, 'yahoo', start, end)

df_stock.head()

df_stock.columns = ['High','Low','Open','Close','Volume_stock','Adj_Close_stock']

df_stock['HiLo_vola_stock'] = (df_stock['High'] - df_stock['Low']) / df_stock['Adj_Close_stock'] *
100.0

df_stock['Pct_change_stock'] = (df_stock['Close'] - df_stock['Open']) / df_stock['Open'] * 100.0

stock_1 = df_stock[['Pct_change_stock']].values.astype(float)

scaler = StandardScaler().fit(stock_1)

scaled_data = scaler.transform(stock_1)

df_stock['Pct_change_stock_scaled'] = scaled_data

df_stock.tail()

df_full =
pd.concat([df_stock[['Volume_stock','Adj_Close_stock','HiLo_vola_stock','Pct_change_stock',
'Pct_change_stock_scaled']],\
                    df_daily_mean], axis=1, sort=False)
df_full.tail()

df_full['Favs'].fillna(df_full['Favs'].mean(), inplace=True)
df_full['RTs'].fillna(df_full['RTs'].mean(), inplace=True)
df_full['Followers'].fillna(df_full['Followers'].mean(), inplace=True)
```

```
df_full['Following'].fillna(df_full['Following'].mean(), inplace=True)
df_full['Is a RT'].fillna(df_full['Is a RT'].mean(), inplace=True)
df_full['compound'].fillna(df_full['compound'].mean(), inplace=True)
df_full['neg'].fillna(df_full['neg'].mean(), inplace=True)
df_full['neu'].fillna(df_full['neu'].mean(), inplace=True)
df_full['pos'].fillna(df_full['pos'].mean(), inplace=True)
df_full['Compound_multiplied'].fillna(df_full['Compound_multiplied'].mean(), inplace=True)
df_full['Compound_multiplied_scaled'].fillna(df_full['Compound_multiplied_scaled'].mean(),
inplace=True)

df_full.tail()

df_full[[ "Volume_stock", "Adj_Close_stock", "HiLo_vola_stock", "Pct_change_stock",
"Pct_change_stock_scaled"]] = \
df_full[[ "Volume_stock", "Adj_Close_stock", "HiLo_vola_stock", "Pct_change_stock",
"Pct_change_stock_scaled"]] \
.interpolate(method='linear', limit_direction='forward', axis=0)

df_full.tail(11)

pd.DataFrame.describe(df_full)


import math

forecast_col = 'Pct_change_stock'

forecast_out = int(math.ceil(0.013 * len(df_full)))

buy_or_sell = []

for row in df_full['Pct_change_stock']:
    if row >= 0:
        buy_or_sell.append(1)
    elif row < 0:
        buy_or_sell.append(-1)

#Adds -1 or +1 to the column based on if 'Predicted_change' is negative or positive
df_full['Buy/Sell'] = buy_or_sell

# The 'Buy/Sell' values need to be shifted up one row to match the 'Predicted_change' values
df_full['Buy/Sell'] = df_full['Buy/Sell'].shift(-1)
```

```
df_full.head()

pd.DataFrame.describe(df_full)

fig_size = plt.rcParams["figure.figsize"]
fig_size[0] =16.0
fig_size[1] = 4.0

x = df_full['Compound_multiplied_scaled']
plt.hist(x, normed=True, bins=250)
plt.ylabel('Compound_multiplied_scaled')

writer_df = pd.ExcelWriter('df_full.xlsx', engine='xlsxwriter')
df_full.to_excel(writer_df)
writer_df.save()

os.rename('df_full.xlsx', '$AAL.xlsx') # UPDATE THE $CASHTAG BEFORE RUNNING THE
CELL!!!!
df_full['Predicted_change_stock'] = df_full[forecast_col].shift(-forecast_out)


# Training the binary classifiers

%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
import os
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.naive_bayes import MultinomialNB
from sklearn.preprocessing import MinMaxScaler
from sklearn import tree
import pydotplus
from sklearn.externals.six import StringIO
from IPython.display import Image
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score
import xlsxwriter
```

```
import os

import glob

glob.glob('/Users/yuechenjiang/Desktop/FE690/Tweets/*.xlsx')

all_data = pd.DataFrame()
for f in glob.glob('/Users/yuechenjiang/Desktop/FE690/Tweets/*.xlsx'):
    df = pd.read_excel(f)
    all_data = all_data.append(df,ignore_index=True)

xls = pd.ExcelFile('/Users/yuechenjiang/Desktop/FE690/Tweets/$AAL_minusones.xlsx') # Update
cashtag; separate for each run.
all_data = pd.read_excel(xls, header = 0,encoding='latin-1')

all_data = all_data[np.isfinite(all_data['Predicted_change_stock'])]
all_data = all_data[np.isfinite(all_data['Buy/Sell'])]

nan_rows = all_data[all_data['Predicted_change_stock'].isnull()]
nan_rows

len(all_data)
# 74

all_data.describe()

all_data.info()

fig_size = plt.rcParams["figure.figsize"]
fig_size[0] =16.0
fig_size[1] = 4.0
x = all_data['Compound_multiplied_scaled']
plt.hist(x, normed=True, bins=250)
plt.ylabel('Compound_multiplied_scaled')

x = np.array(all_data[['Compound_multiplied_scaled']])
y = np.array(all_data['Buy/Sell'])

# from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 42)

# KNN K-Nearest-Neughbors
```

```
# Accuracy from running only 20% Testing data against 80% Training data

from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score

neigh = KNeighborsClassifier(n_neighbors=5)
neigh.fit(x_train, y_train)
neigh.score( x_test, y_test)
# 0.59999999999999998

# Accuracy after cross valldation
neigh_cv = cross_val_score(neigh, x_train, y_train, cv=10)
print(neigh_cv.mean())
# 0.575238095238

# Logistic Regression
from sklearn.linear_model import LogisticRegression

logreg = LogisticRegression(random_state=42)
logreg.fit(x_train, y_train)
logreg.score( x_test, y_test)
# 0.59999999999999998

# Accuracy after cross valldation
logreg_cv = cross_val_score(logreg, x_train, y_train, cv=10)
print(logreg_cv.mean())
# 0.627619047619

# Support Vector Machines (SVM) with linear kernel¶
# kernel = 'linear'
from sklearn.svm import SVC

svm_linear = SVC( kernel = 'linear')
svm_linear.fit(x_train, y_train)
svm_linear.score(x_test, y_test)
# 0.59999999999999998

# Accuracy for 'linear' after cross validation
svm_linear_cv = cross_val_score(svm_linear, x_train, y_train, cv=10)
print(svm_linear_cv.mean())
# 0.627619047619

# kernel = 'rbf' - NOT USED
```

```
# svm_rbf = SVC( kernel = 'rbf')
# svm_rbf.fit(x_train, y_train)
# Accuracy for 'rbf' after cross validation
# svm_rbf_cv = cross_val_score(svm_rbf, x_train, y_train, cv=10)
# print(svm_rbf_cv.mean())

# Naive Bayes
# A MinMaxScaler is needed to get the features in the range MultinomialNB requires. No 20/80 testing
was done, only cross-validation.
from sklearn.naive_bayes import MultinomialNB

from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_minmax = scaler.fit_transform(x_train)

mnb = MultinomialNB()

# Accuracy after cross validation
mnb_cv = cross_val_score(mnb, X_minmax, y_train, cv=10) # uscaled data accuracy same;
6588046192259676
print(mnb_cv.mean())
# 0.627619047619

print("KNN: \t\t\t", neigh_cv.mean())
print("Logistic Regression: \t", logreg_cv.mean())
print("SVM linear: \t\t", svm_linear_cv.mean())
print("Naive Bayes: \t\t", mnb_cv.mean())
# Results Summary
# KNN:                          0.575238095238
# Logistic Regression:      0.627619047619
# SVM linear:               0.627619047619
# Naive Bayes:                  0.627619047619

results = []
cv = [neigh_cv.mean(), logreg_cv.mean(), svm_linear_cv.mean(),mnb_cv.mean()]
results.append(cv)

results = {'0': ['KNN', 'LogReg', 'SVM linear', 'Naive Bayes'],
        '1':[neigh_cv.mean(), logreg_cv.mean(), svm_linear_cv.mean(), mnb_cv.mean()]}

summary = pd.DataFrame.from_dict(results)
summary = summary.transpose()
summary = summary.rename(index = {'0':'Model', '1':'AAL'}) # Update cashtag!
```

```
# Save the result from each classifier for this cashtag in an excel
# The results are used for creating an overall summary for each cashtag and classifier.
writer_df = pd.ExcelWriter('summary.xlsx', engine='xlsxwriter')
summary.to_excel(writer_df)
writer_df.save()

os.rename('summary.xlsx', 'AAL_summary.xlsx') # Update cashtag!



# Processing new data for predictions

%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
import datetime as dt
import pandas_datareader.data as web
import math
import xlsxwriter
import os

# Change file- and stockname/cashtag and run all cells
xls = pd.ExcelFile('/Users/yuechenjiang/Desktop/FE690/Tweets/$WDC_3.xlsx') # CHANGE FILE
NAME!!!

stock = "WDC" #CHANGE STOCK TICKER!!!
df = pd.read_excel(xls, header = 0,encoding='latin-1')

df.head()

# Add column with just the date, remove column with date & time, rearrange columns
df["date"] = df["created at"].dt.date
df.tail()

# Add sentiment to each tweet using Vader
Tweet = df['text']
Tweet.head()
```

```
def sentimentScore(Tweet):
    analyzer = SentimentIntensityAnalyzer()
    results = []
    for sentence in Tweet:
        vs = analyzer.polarity_scores(sentence)
        print("Vader score: " + str(vs))
        results.append(vs)
    return results

df_results = pd.DataFrame(sentimentScore(Tweet))

# Combining the two dataframes
df_results.head()

df_tweets = pd.merge(df, df_results, left_index=True, right_index=True)

df_tweets.tail()

# Converting 'date' column from object to datetime
df_tweets['date'] = pd.to_datetime(df_tweets['date'])

# Choose the common range for the dataframes to be used for all tweet data
df_tweets = df_tweets[(df_tweets['date'] >= '2019-02-28') & (df_tweets['date'] <= '2019-03-28')]

df_tweets.tail()

# Adding a datetime column/index
df_tweets['datetime'] = pd.to_datetime(df_tweets['date']) # change of created at column to datetime
columns
df_tweet_SA = df_tweets.set_index('datetime') # creates a new dataframe 'df_tweet_SA' with the new
index column datetime
df_tweet_SA.drop(['date'], axis=1, inplace=True) #drops the original 'created at' column from the
dataframe
df_tweet_SA.tail()

# Slimming down the stream into a dataframe with only relevant columns.
df_tweet_SA = df_tweets[['datetime', 'text', 'follower count', 'compound','neg','neu','pos']]

df_tweet_SA.head()

# Remove tweets were compound is zero
df_tweet_SA = df_tweet_SA[(df_tweet_SA[['compound']] != 0).all(axis=1)]
```

```
# Create new column with the 'compound' multiplied by nr of followers of the Tweeter
df_tweet_SA['Compound_multiplied'] = df_tweet_SA['compound']*df_tweet_SA['follower count']
df_tweet_SA.head()

# Remove rows where 'follower count' is NaN
nan_rows = df_tweet_SA[df_tweet_SA['follower count'].isnull()]
nan_rows

df_tweet_SA = df_tweet_SA[np.isfinite(df_tweet_SA['follower count'])]

# Creat a df with daily MEANS of each column
df_daily_mean=(df_tweet_SA.groupby(df_tweet_SA.datetime).mean())

df_daily_mean.tail()

len(df_daily_mean)
# 28

# Downloading stock data from Yahoo Finance
import pandas_datareader.data as web

start = dt.datetime(2020, 2, 28)
end =  dt.datetime(2020, 3, 28) #dt.datetime.now()

df_stock = web.DataReader(stock, 'yahoo', start, end)

df_stock.tail()

# New column for daily percent change - stock
df_stock['Pct_change'] = (df_stock['Close'] - df_stock['Open']) / df_stock['Open'] * 100.0
df_stock.tail()

# Combine the tweet sentiment dataframe with the stock data dataframe
df_full = pd.concat([df_stock[['High', 'Low', 'Open', 'Adj Close', 'Pct_change']],\
                    df_daily_mean], axis=1, sort=False)
df_full.tail(11)

df_full['follower count'].fillna(df_full['follower count'].mean(), inplace=True)
df_full['compound'].fillna(df_full['compound'].mean(), inplace=True)
df_full['neg'].fillna(df_full['neg'].mean(), inplace=True)
df_full['neu'].fillna(df_full['neu'].mean(), inplace=True)
df_full['pos'].fillna(df_full['pos'].mean(), inplace=True)
df_full['Compound_multiplied'].fillna(df_full['Compound_multiplied'].mean(), inplace=True)
```

```
df_full.head()

# Interpolate for missing weekend stock data
df_full = df_full[[ 'High', 'Low', 'Open', 'Adj Close', 'Pct_change', 'follower count', 'compound','neg',
'neu', 'pos',\
        'Compound_multiplied' ]].interpolate(method='linear', limit_direction='forward', axis=0)

df_full.tail(22)

pd.DataFrame.describe(df_full)

len(df_full)
# 30

import math

forecast_col = 'Pct_change'

forecast_out = int(math.ceil(0.0333 * len(df_full)))

df_full['Predicted_change'] = df_full[forecast_col].shift(-forecast_out)

buy_or_sell = []

for row in df_full['Pct_change']:
    if row >= 0:
        buy_or_sell.append(1)
    elif row < 0:
        buy_or_sell.append(-1)

#Adds minus 1 or 1 to the column based on if 'Predicted_change' is negative or positive
df_full['Buy/Sell'] = buy_or_sell

# The 'Buy/Sell' values need to be shifte up on row to match the 'Predicted_change' values
df_full['Buy/Sell'] = df_full['Buy/Sell'].shift(-1)

df_full.head(50)

writer_df = pd.ExcelWriter('df_full.xlsx', engine='xlsxwriter')
df_full.to_excel(writer_df)
writer_df.save()
```

```
os.rename('df_full.xlsx', '$WDC_prediction.xlsx') # UPDATE THE $CASHTAG BEFORE RUNNING THE CELL!!!!

# Model outcome & Market

%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
import os
import xlsxwriter

xls = pd.ExcelFile('/Users/yuechenjiang/Desktop/FE690/Tweets/$AAL_prediction.xlsx') # Update cashtag!
Spring2020 = pd.read_excel(xls, header = 0,encoding='latin-1')

Spring2020.tail(14)

Spring2020['Buy/Sell'] = Spring2020['Buy/Sell'].replace(0, -1)

nan_rows = Spring2020[Spring2020['Predicted_change'].isnull()]
nan_rows

# Remove the last row for each cashtag, since its 'Predicted_change' is NaN
Spring2020 = Spring2020[np.isfinite(Spring2019['Predicted_change'])]
Spring2020 = Spring2020[np.isfinite(Spring2019['Buy/Sell'])]
Spring2020.tail()
Spring2020.describe()

xls = pd.ExcelFile('/Users/yuechenjiang/Desktop/FE690/Tweets/$AAL_minusones.xlsx')

all_data = pd.read_excel(xls, header = 0,encoding='latin-1')

# Remove the last row for each cashtag, since its 'Predicted_change_stock' is NaN
all_data = all_data[np.isfinite(all_data['Predicted_change_stock'])]
all_data = all_data[np.isfinite(all_data['Buy/Sell'])]

x_train = np.array(all_data[['Compound_multiplied']])
y_train = np.array(all_data[['Buy/Sell']])

x_test = np.array(Spring2020[['Compound_multiplied']])
```

```
y_test = np.array(Spring2020[['Buy/Sell']])

# KNN - K-Nearest-Neighbors
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score

neigh = KNeighborsClassifier(n_neighbors=7)
neigh.fit(x_train, y_train.ravel())

neigh_cv = cross_val_score(neigh, x_train, y_train.ravel(), cv=10) #Also here, and below, -- .ravel()--
was needed
print(neigh_cv.mean())
# 0.628571428571

# Logistic Regression
from sklearn.linear_model import LogisticRegression

logreg = LogisticRegression(random_state=42)

logreg.fit(x_train, y_train.ravel())

logreg_cv = cross_val_score(logreg, x_train, y_train.ravel(), cv=10)
print(logreg_cv.mean())
# 0.605952380952

# Support Vector Machines (SVM) with different kernels
from sklearn.svm import SVC

svm_linear = SVC( kernel = 'linear')
svm_linear.fit(x_train, y_train.ravel())

svm_linear_cv = cross_val_score(svm_linear, x_train, y_train.ravel(), cv=10)
print(svm_linear_cv.mean())
# 0.618452380952

# Naive Bayes
from sklearn.naive_bayes import MultinomialNB

from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_minmax = scaler.fit_transform(x_train)

mnb = MultinomialNB()
```

```
mnb_cv = cross_val_score(mnb, X_minmax, y_train.ravel(), cv=10)
print(mnb_cv.mean())
# 0.622619047619

print("KNN: \t\t\t", neigh_cv.mean())
print("Logistic Regression: \t", logreg_cv.mean())
print("SVM linear: \t\t", svm_linear_cv.mean())
print("Naive Bayes: \t\t", mnb_cv.mean())
# KNN:                          0.628571428571
# Logistic Regression:          0.605952380952
# SVM linear:                   0.618452380952
# Naive Bayes:                  0.622619047619

results = []
cv = [neigh_cv.mean(), logreg_cv.mean(), svm_linear_cv.mean(),mnb_cv.mean()]
results.append(cv)

results = {'0': ['KNN', 'LogReg', 'SVM linear', 'Naive Bayes'],
        '1':[neigh_cv.mean(), logreg_cv.mean(), svm_linear_cv.mean(),mnb_cv.mean()]}

summary = pd.DataFrame.from_dict(results)
summary = summary.transpose()
summary = summary.rename(index = {'0':'Model', '1':'AAL'}) # Update cashtag!

writer_df = pd.ExcelWriter('summary.xlsx', engine='xlsxwriter')
summary.to_excel(writer_df)
writer_df.save()

os.rename('summary.xlsx', 'AAL_summary.xlsx') # Update cashtag!
Spring2020.tail()

# Add new columns of Spring20 for each of the six ML models' prediction
X = all_data.iloc[:, 14:15].values # creates the 'Compound_multiplied' values from all_data dataframe
as a numpy.ndarray
y = all_data['Buy/Sell']

Buy_or_Sell_KNN = []
Buy_or_Sell = neigh.fit(X, y)
outcome_KNN = (Buy_or_Sell.predict(Spring2019[['Compound_multiplied']]))
Spring2020['KNN_prediction'] = outcome_KNN

Buy_or_Sell_LogReg = []
```

```
Buy_or_Sell = logreg.fit(X, y)
outcome_logreg = (Buy_or_Sell.predict(Spring2019[['Compound_multiplied']]))
Spring2020['LogReg_prediction'] = outcome_logreg

Buy_or_Sell_SVM = []
Buy_or_Sell = neigh.fit(X, y)
outcome_svm = (Buy_or_Sell.predict(Spring2019[['Compound_multiplied']]))
Spring2020['SVM_prediction'] = outcome_svm

Buy_or_Sell_NB = []
X_minmax = scaler.fit_transform(X) #Is thi sneeded??
Buy_or_Sell = mnb.fit(X_minmax, y)    #mnb.fit(X_minmax, y)
outcome_nb = (Buy_or_Sell.predict(Spring2019[['Compound_multiplied']]))
Spring2020['Naive_Bayes_prediction'] = outcome_nb

Spring2020.head(11)

Spring2020["Gain_or_Loss_KNN"] = (Spring2019['Adj Close'] -
Spring2019['Open'])*Spring2019['KNN_prediction']
Spring2020["Gain_or_Loss_LogReg"] = (Spring2019['Adj Close'] -
Spring2019['Open'])*Spring2019['LogReg_prediction']
Spring2020["Gain_or_Loss_SVM"] = (Spring2019['Adj Close'] -
Spring2019['Open'])*Spring2019['SVM_prediction']
Spring2020["Gain_or_Loss_NaiveBayes"] = (Spring2019['Adj Close'] -
Spring2019['Open'])*Spring2019['Naive_Bayes_prediction']
Spring2020.head()
Spring2020 = Spring2020.rest_index()

# Takes the 'Adj Close' of the first day and sets it as result for the first day=starting point
first_day_result = Spring2020.iloc[0]['Adj Close']
Spring2020.set_value( 0, 'KNN_Result', first_day_result)
Spring2020.set_value( 0, 'LogReg_Result', first_day_result)
Spring2020.set_value( 0, 'SVM_Result', first_day_result)
Spring2020.set_value( 0, 'Naive_Bayes_Result', first_day_result)

#The cumulative daily result of trading according to the model; B/S or S/B at Open and Close.

for i in range(1, len(Spring2020)):
   Spring2020.loc[i, 'KNN_Result'] = Spring2020.loc[i-1, 'KNN_Result'] + Spring2019.loc[i,
'Gain_or_Loss_KNN']
   Spring2020.loc[i, 'LogReg_Result'] = Spring2020.loc[i-1, 'LogReg_Result'] + Spring2019.loc[i,
'Gain_or_Loss_LogReg']
```

```
    Spring2020.loc[i, 'SVM_Result'] = Spring2020.loc[i-1, 'SVM_Result'] + Spring2019.loc[i, 
'Gain_or_Loss_SVM']
    Spring2020.loc[i, 'Naive_Bayes_Result'] = Spring2020.loc[i-1, 'Naive_Bayes_Result'] + 
Spring2019.loc[i, 'Gain_or_Loss_NaiveBayes']

Spring2020.tail()

%matplotlib inline
import matplotlib.pyplot as plt
import pylab
from pylab import rcParams
plt.style.use('fivethirtyeight') #ggplot or seaborn
plt.rcParams['figure.figsize'] = 25, 22
plt.suptitle('American Airlines (AAL) 28.2.-28.3.2020', fontsize=30) #, verticalalignment='bottom'

ax1 = Spring2020['Adj Close']
ax2 = Spring2020['KNN_Result']
ax3 = Spring2020['LogReg_Result']
ax4 = Spring2020['SVM_Result']
ax5 = Spring2020['Naive_Bayes_Result']

for i in range (1, 5):
    plt.subplots_adjust(hspace=0.6, wspace=0.15)

    plt.subplot(3,2,1)
    plt.plot(ax1, 'r',  linewidth=2)
    plt.plot(ax2,  'b',  linestyle=':', linewidth=5)
    plt.xlabel('Days', fontsize=20)
    plt.ylabel('Price', fontsize=20)
    plt.title('Buy-and-hold vs. KNN strategy', fontsize=25)
    a='Buy-and-hold'
    b='KNN'
    plt.legend((a,b), fontsize=20, loc='upper center', bbox_to_anchor=(0.5, -0.15), ncol=2, 
frameon=True)

    plt.subplot(3,2,2)
    plt.plot(ax1, 'r',  linewidth=2 )
    plt.plot(ax3, 'g',  linestyle=':', linewidth=5)
    plt.xlabel('Days', fontsize=20)
    plt.ylabel('Price', fontsize=20)
    plt.title('Buy-and-hold vs. LogReg strategy', fontsize=25)
    a='Buy-and-hold'
    b='LogReg'
```

```
    plt.legend((a,b), fontsize=20, loc='upper center', bbox_to_anchor=(0.5, -0.15), ncol=2,
frameon=True)

    plt.subplot(3,2,3)
    plt.plot(ax1, 'r',  linewidth=2 )
    plt.plot(ax4, 'c',  linestyle=':', linewidth=5)
    plt.xlabel('Days', fontsize=20)
    plt.ylabel('Price', fontsize=20)
    plt.title('Buy-and-hold vs SVM strategy', fontsize=25)
    a='Buy-and-hold'
    b='SVM'
    plt.legend((a,b), fontsize=20, loc='upper center', bbox_to_anchor=(0.5, -0.15), ncol=2,
frameon=True)

    plt.subplot(3,2,4)
    plt.plot(ax1, 'r',  linewidth=2 )
    plt.plot(ax5, 'm',  linestyle=':', linewidth=5)
    plt.xlabel('Days', fontsize=20)
    plt.ylabel('Price', fontsize=20)
    plt.title('Buy-and-hold vs NaiveBayes strategy', fontsize=25)
    a='Buy-and-hold'
    b='NaiveBayes'
    plt.legend((a,b), fontsize=20, loc='upper center', bbox_to_anchor=(0.5, -0.15), ncol=2,
frameon=True)

#pylab.savefig('AAL.jpg') # Saves figure as .jpg-file
plt.show()
```