CMSC498O: Final Report
Predicting the Directional Movement of Stock Prices
Albert Koy & Nadeem Malik

Project repository: https://github.com/akoy93/CMSC498O_Final_Project
Collected/transformed data:
https://github.com/akoy93/CMSC498O_Final_Project/tree/master/data
Model Predictions:
https://github.com/akoy93/CMSC498O_Final_Project/tree/master/data/2014-12-02_1yr_predictions

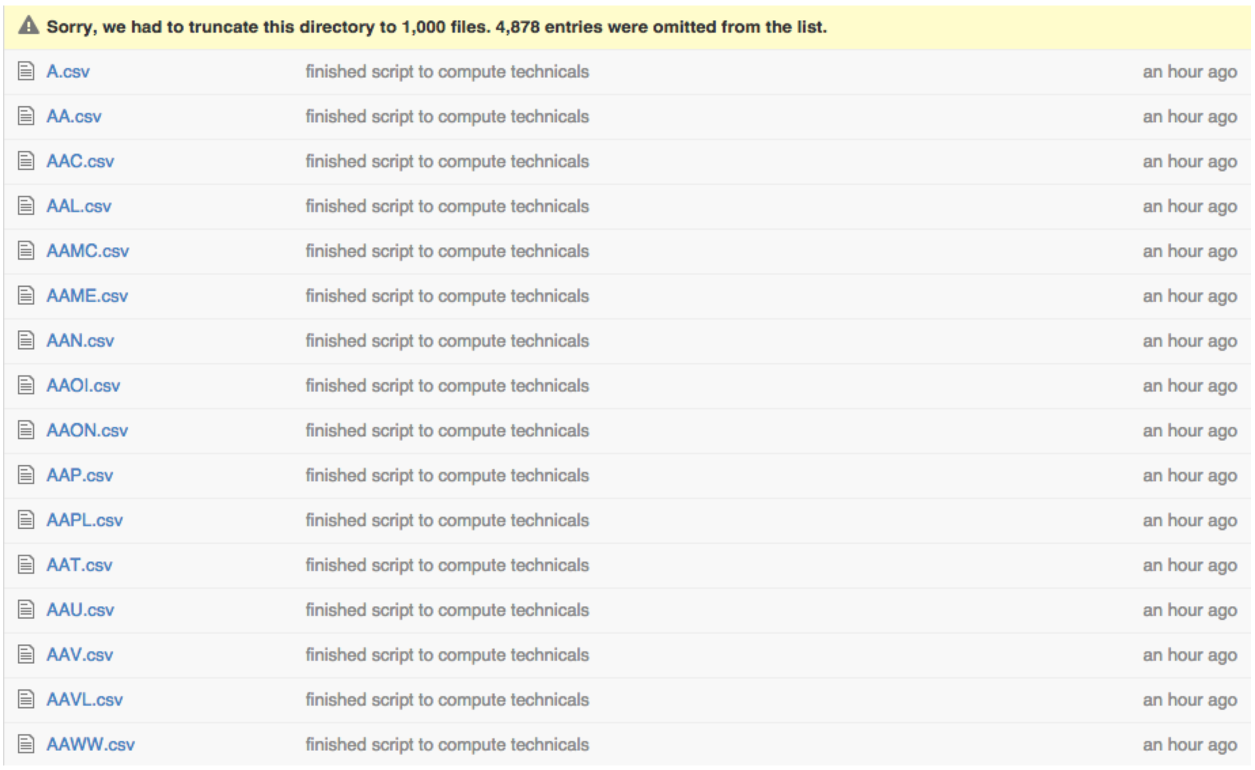
Introduction

For our final project, we created a model to predict the directional movement of a stock price given historical data, what is known as technical analysis. The goal of our project is to predict whether a stock will close higher than it opened given 10 prior days of stock data. We applied machine learning algorithms to historical stock data and we found that by utilizing logistic regression, we were able to predict correctly with ~81% accuracy and ~86% AUC with cross-validation. Our intermediate report focused on the research and survey of commonly used machine learning techniques for technical analysis, which led us to use logistic regression. This report will focus on the practical aspect of our project, the process for getting the data, creating the models, and displaying our results.

Process and Model Development

Gathering and Preprocessing Data

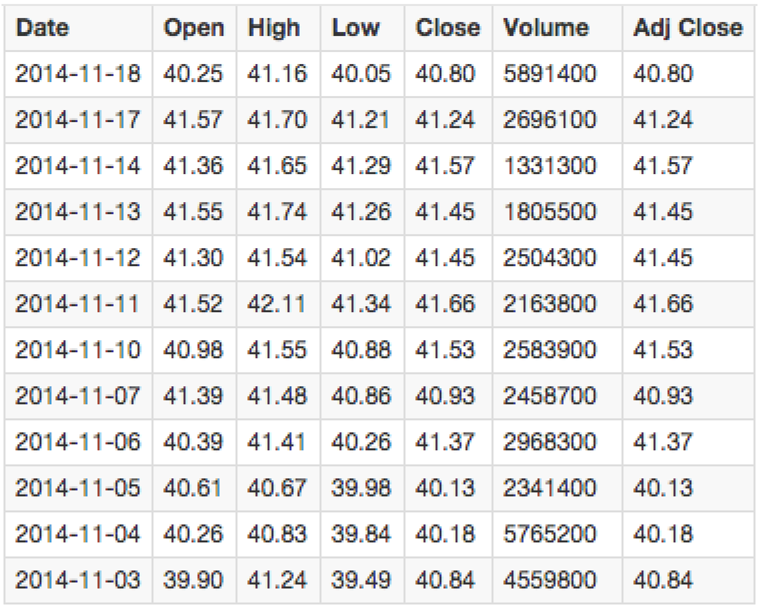
The first challenge in taking on this project was to find a source for historical stock market data. Unfortunately, we couldn't find many large, free, downloadable sets of historical stock market data on the Internet, so we had to write a web scraper to gather this data. We did this by writing a Ruby script to scrape the Yahoo! Finance page to grab data for all U.S. stocks on a daily timeframe over the past 5 years. We now have cleanly formatted CSV files for each U.S. Stock symbol (*Figure 2*) containing the Open, High, Low, Close, and Adjusted Close prices for each stock along with the daily volume (*Figure 3*).



Sorry, we had to truncate this directory to 1,000 files. 4,878 entries were omitted from the list.

A.csv	finished script to compute technicals	an hour ago
AA.csv	finished script to compute technicals	an hour ago
AAC.csv	finished script to compute technicals	an hour ago
AAL.csv	finished script to compute technicals	an hour ago
AAMC.csv	finished script to compute technicals	an hour ago
AAME.csv	finished script to compute technicals	an hour ago
AAN.csv	finished script to compute technicals	an hour ago
AAOI.csv	finished script to compute technicals	an hour ago
AAON.csv	finished script to compute technicals	an hour ago
AAP.csv	finished script to compute technicals	an hour ago
AAPL.csv	finished script to compute technicals	an hour ago
AAT.csv	finished script to compute technicals	an hour ago
AAU.csv	finished script to compute technicals	an hour ago
AAV.csv	finished script to compute technicals	an hour ago
AAVL.csv	finished script to compute technicals	an hour ago
AAWW.csv	finished script to compute technicals	an hour ago

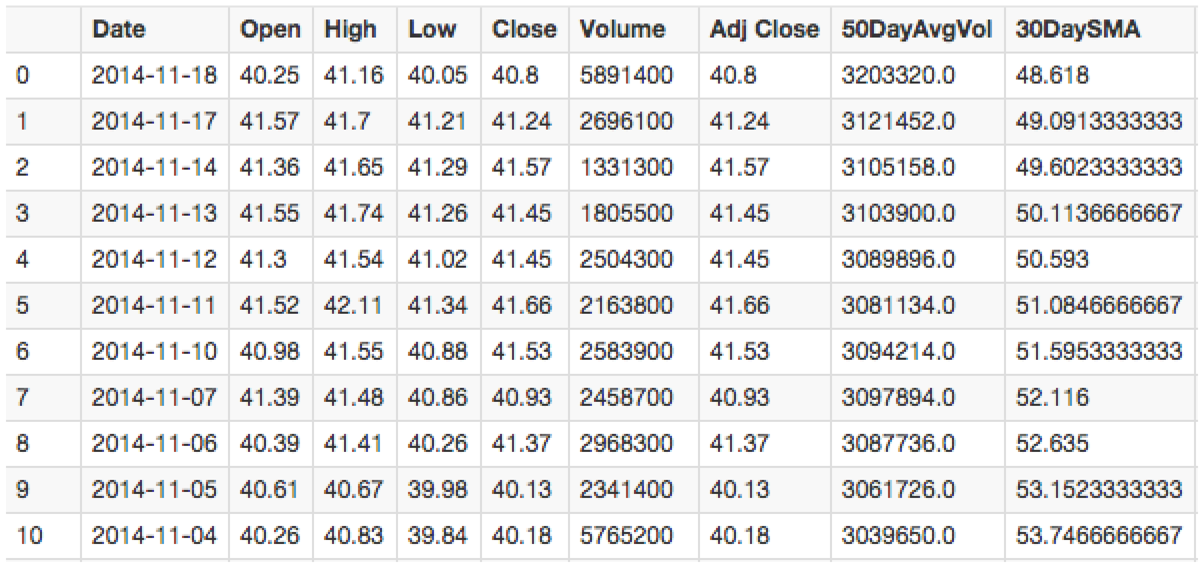
Figure 2



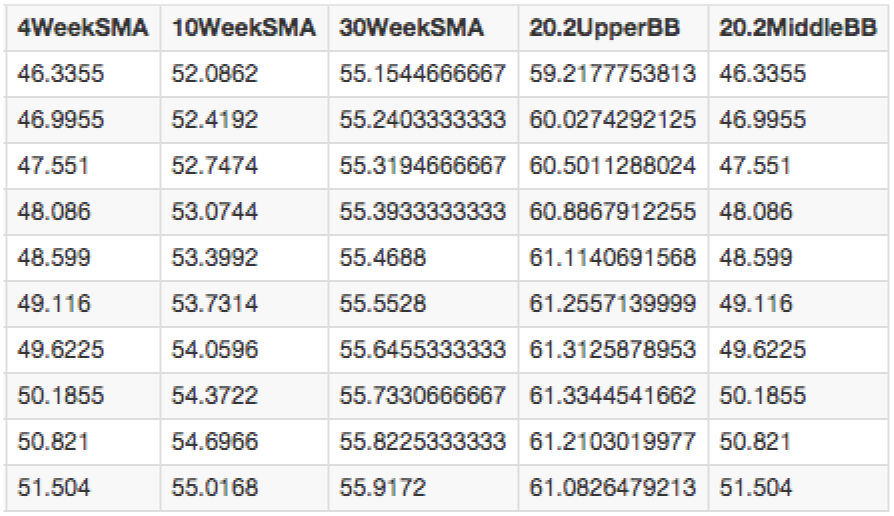
Date	Open	High	Low	Close	Volume	Adj Close
2014-11-18	40.25	41.16	40.05	40.80	5891400	40.80
2014-11-17	41.57	41.70	41.21	41.24	2696100	41.24
2014-11-14	41.36	41.65	41.29	41.57	1331300	41.57
2014-11-13	41.55	41.74	41.26	41.45	1805500	41.45
2014-11-12	41.30	41.54	41.02	41.45	2504300	41.45
2014-11-11	41.52	42.11	41.34	41.66	2163800	41.66
2014-11-10	40.98	41.55	40.88	41.53	2583900	41.53
2014-11-07	41.39	41.48	40.86	40.93	2458700	40.93
2014-11-06	40.39	41.41	40.26	41.37	2968300	41.37
2014-11-05	40.61	40.67	39.98	40.13	2341400	40.13
2014-11-04	40.26	40.83	39.84	40.18	5765200	40.18
2014-11-03	39.90	41.24	39.49	40.84	4559800	40.84

Figure 3

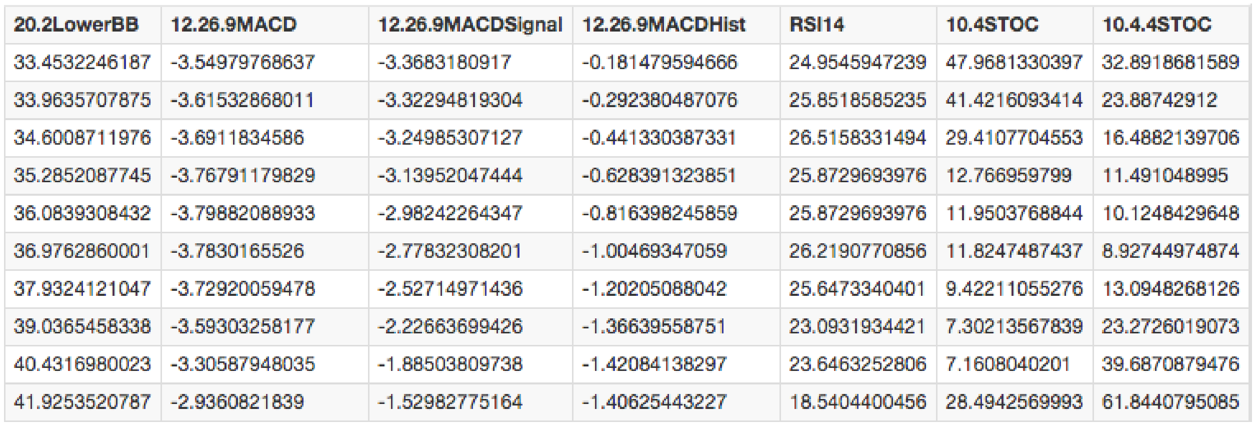
After retrieving the raw stock data, the next step was to compute technical analysis indicators for each stock. This meant writing a script to transform each of the 5000+ CSV files into a file that would give us some more information to work with for each stock. We used Python to write this script and relied heavily on the NumPy, Pandas, and TA-Lib libraries. TA-Lib is an industry standard library for performing technical analysis, with many functions to help compute indicators. To augment the raw data that we retrieved from Yahoo! Finance, we computed the 50 Day Average Volume, 30 Day, 4 Week, 10 Week, and 30 Week Simple Moving Averages, the Daily 20.2 Upper, Middle, and Lower Bollinger Bands, the Daily 12.26.9 MACD, 12.26.9 MACD Signal, and 12.26.9 MACD Histogram values, the Relative Strength Index with 14 bars as the timeframe, and the Daily 10.4 and 10.4.4 Stochastic values.



	Date	Open	High	Low	Close	Volume	Adj Close	50DayAvgVol	30DaySMA
0	2014-11-18	40.25	41.16	40.05	40.8	5891400	40.8	3203320.0	48.618
1	2014-11-17	41.57	41.7	41.21	41.24	2696100	41.24	3121452.0	49.0913333333
2	2014-11-14	41.36	41.65	41.29	41.57	1331300	41.57	3105158.0	49.6023333333
3	2014-11-13	41.55	41.74	41.26	41.45	1805500	41.45	3103900.0	50.1136666667
4	2014-11-12	41.3	41.54	41.02	41.45	2504300	41.45	3089896.0	50.593
5	2014-11-11	41.52	42.11	41.34	41.66	2163800	41.66	3081134.0	51.0846666667
6	2014-11-10	40.98	41.55	40.88	41.53	2583900	41.53	3094214.0	51.5953333333
7	2014-11-07	41.39	41.48	40.86	40.93	2458700	40.93	3097894.0	52.116
8	2014-11-06	40.39	41.41	40.26	41.37	2968300	41.37	3087736.0	52.635
9	2014-11-05	40.61	40.67	39.98	40.13	2341400	40.13	3061726.0	53.1523333333
10	2014-11-04	40.26	40.83	39.84	40.18	5765200	40.18	3039650.0	53.7466666667



4WeekSMA	10WeekSMA	30WeekSMA	20.2UpperBB	20.2MiddleBB
46.3355	52.0862	55.1544666667	59.2177753813	46.3355
46.9955	52.4192	55.2403333333	60.0274292125	46.9955
47.551	52.7474	55.3194666667	60.5011288024	47.551
48.086	53.0744	55.3933333333	60.8867912255	48.086
48.599	53.3992	55.4688	61.1140691568	48.599
49.116	53.7314	55.5528	61.2557139999	49.116
49.6225	54.0596	55.6455333333	61.3125878953	49.6225
50.1855	54.3722	55.7330666667	61.3344541662	50.1855
50.821	54.6966	55.8225333333	61.2103019977	50.821
51.504	55.0168	55.9172	61.0826479213	51.504



20.2LowerBB	12.26.9MACD	12.26.9MACDSignal	12.26.9MACDHist	RSI14	10.4STOC	10.4.4STOC
33.4532246187	-3.54979768637	-3.3683180917	-0.181479594666	24.9545947239	47.9681330397	32.8918681589
33.9635707875	-3.61532868011	-3.32294819304	-0.292380487076	25.8518585235	41.4216093414	23.88742912
34.6008711976	-3.6911834586	-3.24985307127	-0.441330387331	26.5158331494	29.4107704553	16.4882139706
35.2852087745	-3.76791179829	-3.13952047444	-0.628391323851	25.8729693976	12.766959799	11.491048995
36.0839308432	-3.79882088933	-2.98242264347	-0.816398245859	25.8729693976	11.9503768844	10.1248429648
36.9762860001	-3.7830165526	-2.77832308201	-1.00469347059	26.2190770856	11.8247487437	8.92744974874
37.9324121047	-3.72920059478	-2.52714971436	-1.20205088042	25.6473340401	9.42211055276	13.0948268126
39.0365458338	-3.59303258177	-2.22663699426	-1.36639558751	23.0931934421	7.30213567839	23.2726019073
40.4316980023	-3.30587948035	-1.88503809738	-1.42084138297	23.6463252806	7.1608040201	39.6870879476
41.9253520787	-2.9360821839	-1.52982775164	-1.40625443227	18.5404400456	28.4942569993	61.8440795085

Figure 4

Developing Models

Next, once we had our raw stock data, we set out to train three types of models that we identified in our intermediate report: Logistic Regression, SVM with Linear kernel, and Random Forest Classifier. We wanted to be able to input 10 days of data into our model in order to predict the directional movement of the last day's closing price. The biggest challenge in this step was feature selection. Initially, we found that by training on all of the 21 features, we were getting very poor results. In most cases, we were getting predictive accuracy that was essentially the same as flipping a coin for this binary classification problem. Moreover, since we had 5 years of data for ~6000 stocks and 21 features per day, computation time was extremely expensive and we were never able to train our models with a non-trivial set of our raw data.

One major insight that we made that allowed us to speed up our training process was that the technical indicators we calculated were all derivatives of the initial set of raw data that we started with. Thus, we decided to start training on our initial set of data without the technical indicators, which allowed us to develop a model, but still without any luck in increasing predictive accuracy.

The next major insight we made was that was that we had to normalize our data to add consistency to our dataset between different stocks. We decided to normalize the Open, High, Low, and Close prices by dividing by the value of the Open price on the first day in the 10-day window. Additionally, we decided to normalize the Volume by dividing by the 50-day average volume. At this point, we were able to finish training our Logistic Regression model with a large number of stocks, but the SVM and Random Forest Classifiers were still too slow. We were able to achieve decent accuracy with our Logistic Regression model, predicting ~70% of outcomes correctly with a random sample of data which was roughly half of our dataset.

In order to get a more representative dataset, we observed that it would be worthwhile to train our models by using a market index, such as the S&P 500. The S&P 500 is commonly used to evaluate the general stock market trend, as it is believed to be one of the most representative stock indexes of the entire stock market. By using the S&P 500 as our training set, we were able to achieve a small boost in predictive accuracy, and since this was less data than we were using originally, we were able to finish training our SVM and Random Forest models.

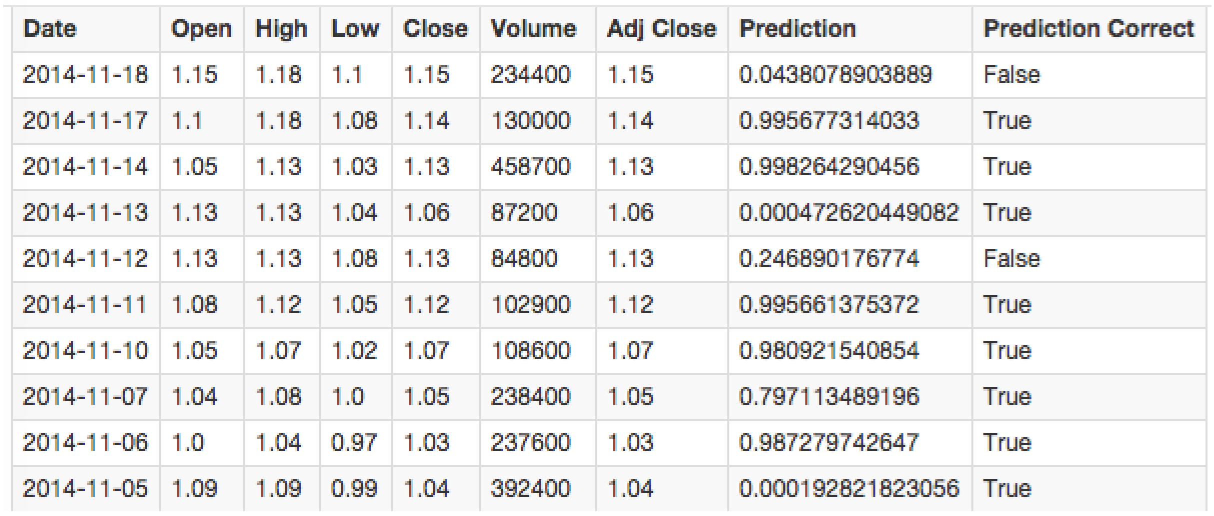
The last insight we made was inspired by a talk given in the University of Maryland's Stock Market Class (HONR348M). According to one guest speaker in that class who makes his living trading stocks in the stock market, he uses only the price action of a stock and draws trend lines between price bars in order to determine whether or not to enter a stock. He mentioned that he didn't use trading volume as an indicator, which is very important according to most expert traders. With that in mind, we decided to train our model without using normalized volume as a feature, and we achieved a ~86% AUC on 5-fold cross-validation when training on 5 years of the S&P 500.

Finally, with more testing, we found that including the High and Low prices in our training data did not make much of a difference in predictive accuracy, so we excluded them from our training set in order to speed up the training process. We ended up with a Logistic Regression model that takes in 19 points of data, the Open and Close for the previous 10 days except for the Close on the last day, and predicts whether the next Close will be higher than the observed Open. We observed that the Linear SVM was slower to train as well as slightly less accurate, and the Random Forest Classifier was nowhere near as accurate, so we decided to use the Logistic Regression method for our final model.

Predictions

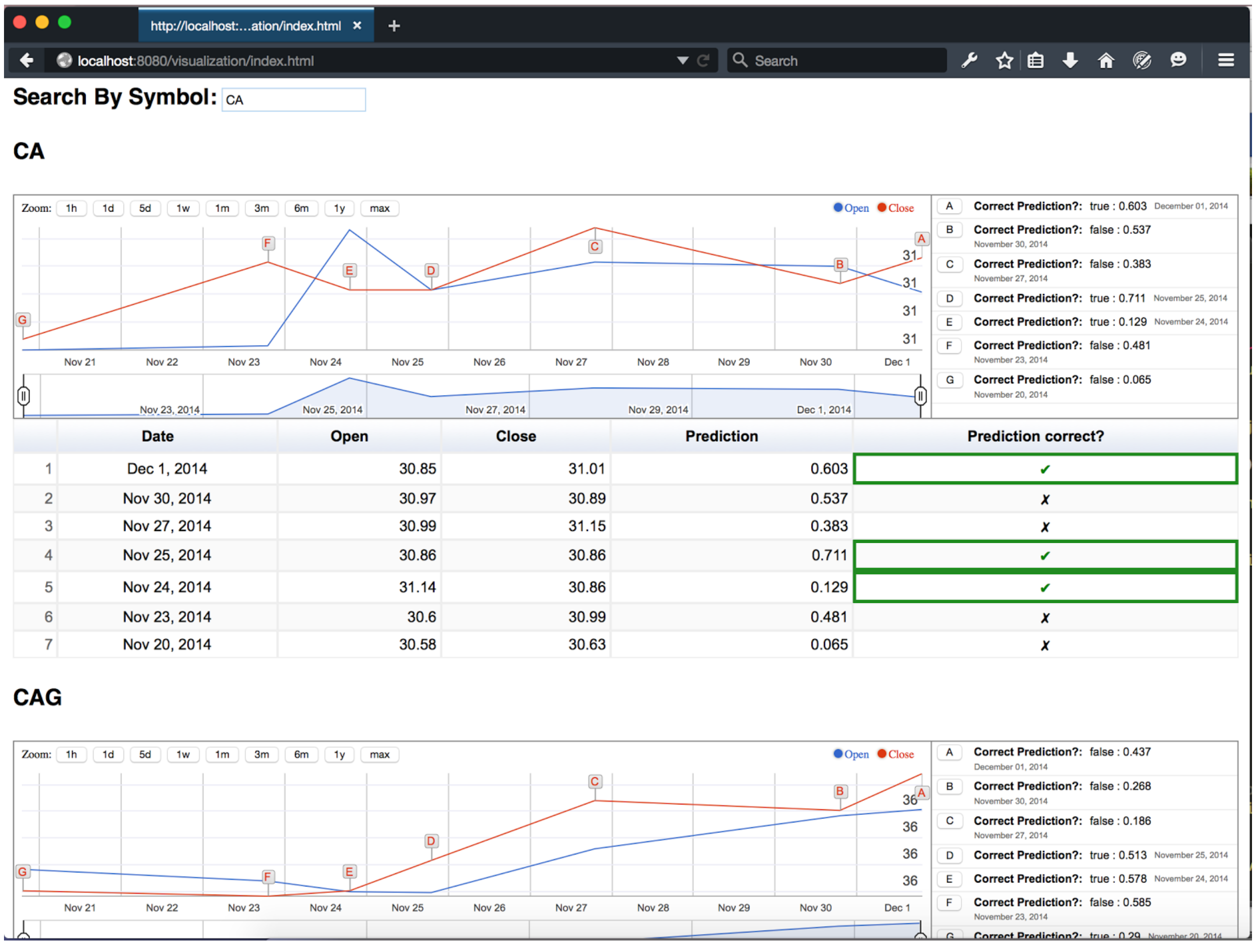
A set of all prediction files is available here:
https://github.com/akoy93/CMSC498O_Final_Project/tree/master/data/2014-12-02_1yr_predictions

We were able to achieve 91% AUC with cross validation when training our Logistic Regression model with S&P 500 stock data over 5 years. This translated to ~81% correctness when making predictions across all ~6000 U.S. stocks. Below is a sample prediction file that is output by our model. It includes "Prediction" and "Prediction Correct" columns appended to the raw stock data which, respectively, give a probability from 0 to 1 of whether the stock will Close higher than the open and a boolean value indicating whether or not the prediction was correct. For our purposes, a value over 0.5 indicates a stock will Close higher, a value below 0.5 indicates a stock will close lower.



Date	Open	High	Low	Close	Volume	Adj Close	Prediction	Prediction Correct
2014-11-18	1.15	1.18	1.1	1.15	234400	1.15	0.0438078903889	False
2014-11-17	1.1	1.18	1.08	1.14	130000	1.14	0.995677314033	True
2014-11-14	1.05	1.13	1.03	1.13	458700	1.13	0.998264290456	True
2014-11-13	1.13	1.13	1.04	1.06	87200	1.06	0.000472620449082	True
2014-11-12	1.13	1.13	1.08	1.13	84800	1.13	0.246890176774	False
2014-11-11	1.08	1.12	1.05	1.12	102900	1.12	0.995661375372	True
2014-11-10	1.05	1.07	1.02	1.07	108600	1.07	0.980921540854	True
2014-11-07	1.04	1.08	1.0	1.05	238400	1.05	0.797113489196	True
2014-11-06	1.0	1.04	0.97	1.03	237600	1.03	0.987279742647	True
2014-11-05	1.09	1.09	0.99	1.04	392400	1.04	0.000192821823056	True

Visualization



Search By Symbol: CA

CA

A	Correct Prediction?: true : 0.603 December 01, 2014
B	Correct Prediction?: false : 0.537 November 30, 2014
C	Correct Prediction?: false : 0.383 November 27, 2014
D	Correct Prediction?: true : 0.711 November 25, 2014
E	Correct Prediction?: true : 0.129 November 24, 2014
F	Correct Prediction?: false : 0.481 November 23, 2014
G	Correct Prediction?: false : 0.065 November 20, 2014

	Date	Open	Close	Prediction	Prediction correct?
1	Dec 1, 2014	30.85	31.01	0.603	✔
2	Nov 30, 2014	30.97	30.89	0.537	✗
3	Nov 27, 2014	30.99	31.15	0.383	✗
4	Nov 25, 2014	30.86	30.86	0.711	✔
5	Nov 24, 2014	31.14	30.86	0.129	✔
6	Nov 23, 2014	30.6	30.99	0.481	✗
7	Nov 20, 2014	30.58	30.63	0.065	✗

CAG

A	Correct Prediction?: false : 0.437 December 01, 2014
B	Correct Prediction?: false : 0.268 November 30, 2014
C	Correct Prediction?: false : 0.186 November 27, 2014
D	Correct Prediction?: true : 0.513 November 25, 2014
E	Correct Prediction?: true : 0.578 November 24, 2014
F	Correct Prediction?: false : 0.585 November 23, 2014
G	Correct Prediction?: true : 0.29 November 20, 2014

(Screenshot shows old prediction data. Newer data is more accurate.)

We used d3.js and Google Visualization API to create a visualization demo that shows the results of our application. The demo displays a series of stocks with corresponding tables and charts. The table lists the open and close prices by date, our prediction estimate, and whether our prediction was correct for that day or not. The chart for each stock shows annotations for when our predictions were correct and has controls to adjust the graphic display by range of time selected. To come up with this visualization, we first read stock symbols from a text file. For each stock symbol, we read it's corresponding CSV file to get a certain number of entries to display. Next, we created a table and chart element in our web page, for each set of entries. There were several challenges we faced when writing code for visualizations. For example, when we tried to use the d3.js library to parse in CSV file for each stock symbol, it was impossible to load all the files as any browser we tried would crash due to insufficient resources. We learned that data visualization has significant resources challenges that need to be realized before one comes up with a demo. On another note, we found the Google Visualization API to be very simple to use and quite elegant. This API uses a DataTable object to store data that is read in from JavaScript arrays, which is then used to create the chart or table you would like to display.

Our Experiences

When initially starting this project, we believed that most of our efforts would be spent choosing different machine learning algorithms and configurations as well as computing our own custom indicators to increase our predictive accuracy. This turned out to not be the case at all. In reality, the vast majority of our time was spent gathering data, writing scripts to process the data, and figuring out how to reduce the amount of data that we needed in order to train the models we used for our project. Given the limited hardware we have, we were required to abandon the vast majority of our data when training or models, which was especially challenging to decide. The actual machine learning implementations required ~10 lines of code by using the SK-Learn Python library, which was very surprising to us. In addition, we realized that visualizations can be more complicated than they seem, even with the vast amount of APIs that make it seem easy, there are other challenges that need to be considered, such as graphical rendering and performance. If we had more time, we would be able to further optimize our visualization for better performance. Overall, this project was a great experience in exploring the realm of practical machine learning, and we hope to have further opportunities involving such work in the future.

Code:

Scraper - get_stocks.rb (Scraping raw data from Yahoo! Finance)

```
require 'date'
require 'open-uri'
```

```
# usage ruby get_stocks.rb #{STOCK_SYMBOLS} #{NUMBER_OF_YEARS}


unless ARGV.length == 2
 puts "Incorrect Usage. Use: ruby get_stocks.rb {STOCK_SYMBOLS_FILE} {NUMBER_OF_YEARS}"
else
 current_date = Date.parse(Time.now.to_s)
 filename = ARGV[0]
 num_years = ARGV[1].to_i
 directory = "#{current_date.to_s}_#{num_years}yr"
 symbols = []


 # remove existing directory
 system("rm -rf #{directory}")
 system("mkdir #{directory}")


 # read stock symbols from file
 File.foreach(filename) do |sym|
  sym.chomp!
  puts "Fetching #{sym}..."
  # api url for stock data retrieval
  retrieval_url = "http://real-chart.finance.yahoo.com/table.csv?s=#{sym}" +
   "&a=#{current_date.month - 1}&b=#{current_date.day}&c=#{current_date.year -
num_years}&d=#{current_date.month - 1}" +
   "&e=#{current_date.day}&f=#{current_date.year}&g=d&ignore=.csv"
  begin
   open("#{directory}/#{sym}.csv", "wb") do |file|
    file << open(retrieval_url).read
   end
   symbols << sym
  rescue
   puts "Unable to retrieve #{sym}"
  end
 end
 File.open("#{directory}/symbol_list.txt", "wb") do |file|
  symbols.each { |sym| file.puts sym }
 end


 puts "DONE"
end
```

Prediction - predict.py (Using our model to output predictions)

```
# Usage: python generate_training.py {DIRECTORY} {MODEL_FILE_NAME}
import numpy as np
import pandas as pd
import glob
import sys
import pickle
from subprocess import call


NUM_DAYS_IN_WINDOW = 10
NUM_POINTS_PER_DAY = 2


if len(sys.argv) != 3:
 print "Incorrect usage. Use: \"python generate_training.py {DIRECTORY} {MODEL_FILE_NAME}\""
 sys.exit()


directory = sys.argv[1]
new_directory = "%s_predictions" % directory
model_file_name = sys.argv[2]
```

```
# create new directory
call(["rm", "-rf", new_directory])
call(["mkdir", new_directory])


with open(model_file_name, "rb") as f:
 model = pickle.load(f)
 print "Model loaded."


 # get all csv files in the given directory
 files = glob.glob("%s/*.csv" % directory)
 file_num = 0


 for f in files:
  file_num += 1
  print "(%d/%d) - Making predictions for %s..." % (file_num, len(files), f[len(directory) + 1:])
  try:
   data = pd.read_csv(f)[::-1]
   data = data.set_index('Date')
   data['Prediction'] = np.nan
   data['Prediction Correct'] = np.nan


   # initialize window
   window = [data.loc[[0]]['Open']]

   i = 0
   for i in range(1, NUM_DAYS_IN_WINDOW):
    row = data.loc[[i]]
    window = [row['Open'], row['Close']] + window
    # normalize
    window_arr = np.hstack(np.asarray(window))
    normalized = window_arr / window_arr[0]


    # result
    idx = data.loc[[0]].index
    data.loc[idx, 'Prediction'] = model.predict_proba([normalized])[:,1]
    data.loc[idx, 'Prediction Correct'] = bool(model.predict([normalized])[0]) == (data.loc[idx, 'Close'] >=
data.loc[idx, 'Open'])


    while i < data.shape[0] - 1:
     i += 1
     row = data.loc[[i]]
     window_arr = window_arr[:-NUM_POINTS_PER_DAY]
     window_arr = np.hstack([np.hstack([row['Open'], row['Close']]), window_arr])
     normalized = window_arr / window_arr[0]
     idx = data.loc[[i - NUM_DAYS_IN_WINDOW + 1]].index
     data.loc[idx, 'Prediction'] = model.predict_proba([normalized])[:,1]
     data.loc[idx, 'Prediction Correct'] = bool(model.predict([normalized])[0]) == (data.loc[idx, 'Close'] >=
data.loc[idx, 'Open'])


    data = data.dropna()[::-1]
    data.to_csv(f.replace(directory, new_directory), sep=',', encoding='utf-8')
  except:
    print "Unable to make predictions for %s." % f[len(directory) + 1:]
```

Python - generate_training.py (Generating formatted training data to use in our model)

```
# Usage: python generate_training.py {NUMBER_OF_DAYS} {NUM_DAYS_IN_WINDOW} {DIRECTORY}
{OUTPUT_FILE}
import numpy as np
import pandas as pd
import glob
import sys


if len(sys.argv) != 5:
 print "Incorrect usage. Use: \"python generate_training.py {NUMBER_OF_DAYS}
{NUM_DAYS_IN_WINDOW} {DIRECTORY} {OUTPUT_FILE}\""
 sys.exit()


num_days_to_train = int(sys.argv[1])
num_days_in_window = int(sys.argv[2])
directory = sys.argv[3]
output_file = sys.argv[4]


# get all csv files in the given directory
files = glob.glob("%s/*.csv" % directory)
i = 0


results = []


# process all files
for f in files:
 i += 1
 print "(%d/%d) - Generating training data from %s..." % (i, len(files), f[len(directory) + 1:])
 try:
   data = pd.read_csv(f)


   if data.shape[0] >= num_days_to_train:
     # drop unnecessary columns and move into one line
     data = data.set_index('Date')
     data = data.drop('Adj Close', 1)
```

```
    data = data.drop('High', 1)
    data = data.drop('Low', 1)
    data = data.drop('Volume', 1)

    # convert into numpy array
    data = np.array(data)
    data = data[0:num_days_to_train]

    result = []
    for r in range(num_days_in_window):
      result.append(data[r:(r + data.shape[0] - num_days_in_window + 1),:])

    results.append(np.hstack(result))
  except:
    print "ERROR: Unable to generate training data from %s" % f[len(directory) + 1:]

results = np.vstack(results)
np.savetxt(output_file, results, delimiter=",")
```

Python - train_models.py (Training a Logistic Regression model)

```
# Usage: python train_models.py {TRAINING_FILE_NAME} {MODEL_OUTPUT_FILE}
# We're training a model to determine if a stock will close above its open on the next day.
# We're given the stock's data for a variable window and its opening price on the last day.

import numpy as np
import sys
import pickle
import sklearn.linear_model as lm
from sklearn import cross_validation


NUMBER_OF_DAYS_IN_WINDOW = 10
NUM_DATA_POINTS_PER_DAY = 2


def tied_rank(x):
 """
 This function is by Ben Hamner and taken from
https://github.com/benhamner/Metrics/blob/master/Python/ml_metrics/auc.py
 Computes the tied rank of elements in x.
 This function computes the tied rank of elements in x.
 Parameters
 ----------
 x : list of numbers, numpy array
 Returns
 -------
 score : list of numbers
        The tied rank f each element in x
 """
 sorted_x = sorted(zip(x,range(len(x))))
 r = [0 for k in x]
 cur_val = sorted_x[0][0]
 last_rank = 0
 for i in range(len(sorted_x)):
   if cur_val != sorted_x[i][0]:
     cur_val = sorted_x[i][0]
     for j in range(last_rank, i):
       r[sorted_x[j][1]] = float(last_rank+1+i)/2.0
     last_rank = i
   if i==len(sorted_x)-1:
     for j in range(last_rank, i+1):
       r[sorted_x[j][1]] = float(last_rank+i+2)/2.0
 return r
```

```
def auc(actual, posterior):
 """
 This function is by Ben Hamner and taken from
https://github.com/benhamner/Metrics/blob/master/Python/ml_metrics/auc.py

 Computes the area under the receiver-operater characteristic (AUC)
 This function computes the AUC error metric for binary classification.
 Parameters
 ----------
 actual : list of binary numbers, numpy array
        The ground truth value
 posterior : same type as actual
          Defines a ranking on the binary numbers, from most likely to
          be positive to least likely to be positive.
 Returns
 -------
 score : double
       The mean squared error between actual and posterior
 """
 r = tied_rank(posterior)
 num_positive = len([0 for x in actual if x==1])
 num_negative = len(actual)-num_positive
 sum_positive = sum([r[i] for i in range(len(r)) if actual[i]==1])
 auc = ((sum_positive - num_positive*(num_positive+1)/2.0) / (num_negative*num_positive))
 return auc


def auc_scorer(estimator, X, y):
 predicted = estimator.predict_proba(X)[:,1]
 return auc(y, predicted)


if len(sys.argv) != 3:
 print "Incorrect usage. Use: \"python train_models.py {TRAINING_FILE_NAME}
{MODEL_OUTPUT_FILE}\""
 sys.exit()



num_days_in_window = NUMBER_OF_DAYS_IN_WINDOW
points_per_day = NUM_DATA_POINTS_PER_DAY
points_in_window = num_days_in_window * points_per_day



# load training data
training = np.genfromtxt(sys.argv[1], delimiter=",")



# normalize
for row in training:
 opening = row[0]
 for i in range(points_in_window):
  row[i] = row[i] / opening



# pull X and y
X = np.array([p[range(points_in_window - 1)] for p in training])
y = (training[:, points_in_window - points_per_day + 1] > training[:, points_in_window - points_per_day]) +
0



# build model and save
model = lm.LogisticRegression(penalty = "l2").fit(X, y)



with open(sys.argv[2], 'wb') as f:
 pickle.dump(model, f)
 print("Model saved.")
```

```
# print cross validation score
cv_score = np.mean(cross_validation.cross_val_score(model, X, y, cv=5, scoring = auc_scorer))
print "Cross Validation Score = %f" % cv_score
```

Python - test_model.py (Testing a our models)

```
import numpy as np
import sys
import pickle



NUMBER_OF_DAYS_IN_WINDOW = 10
NUM_DATA_POINTS_PER_DAY = 2



if len(sys.argv) != 3:
 print "Incorrect usage. Use: \"python test_model.py {MODEL_FILE_NAME} {TESTING_FILE_NAME}\""
 sys.exit()



model_file = sys.argv[1]
test_file = sys.argv[2]
num_days_in_window = NUMBER_OF_DAYS_IN_WINDOW
points_per_day = NUM_DATA_POINTS_PER_DAY
points_in_window = num_days_in_window * points_per_day



# load testing data
testing = np.genfromtxt(test_file, delimiter=",")



# normalize
for row in testing:
 opening = row[0]
 for i in range(points_in_window):
  row[i] = row[i] / opening



# pull X and y
X = np.array([p[range(points_in_window - 1)] for p in testing])
y = (testing[:, points_in_window - points_per_day + 1] > testing[:, points_in_window - points_per_day]) + 0



# test model
with open(model_file, "rb") as f:
 model = pickle.load(f)
 print model.score(X, y)
```

Python - sample.py (Randomly sampling training and testing data)

```
# Usage: python sample.py {NUM_TRAINING} {NUM_TESTING} {INPUT_FILE}
import pandas as pd
import numpy as np
import sys



if len(sys.argv) == 4:
 num_training = int(sys.argv[1])
 num_testing = int(sys.argv[2])
 input_file = sys.argv[3]



 training_file = "training%d_%s" % (num_training, input_file)
 testing_file = "testing%d_%s" % (num_testing, input_file)



 total_rows = num_training + num_testing
 data = pd.read_csv(input_file)
 if data.shape[0] < total_rows:
  print "Not enough rows in input file."
 else:
  # randomly sample rows of input file
  rows = np.random.choice(data.index.values, total_rows)


  training_rows = sorted(rows[:num_training])
  testing_rows = sorted(rows[-num_testing:])


  # grab rows from dataframe
  data = np.asarray(data)
  training = data[training_rows]
  testing = data[testing_rows]


  np.savetxt(training_file, training, delimiter=",")
  np.savetxt(testing_file, testing, delimiter=",")
else:
 print "Incorrect usage. Use: \"python sample.py {NUM_TRAINING} {NUM_TESTING} {INPUT_FILE}\""
```

Visualization - index.html

```
<!DOCTYPE html>
<meta charset="utf-8">
<script src="http://d3js.org/d3.v3.js"></script>
<script src="http://code.jquery.com/jquery-1.8.2.js"></script>
<script src="http://code.jquery.com/ui/1.9.0/jquery-ui.js"></script>
<link href="http://code.jquery.com/ui/1.9.0/themes/cupertino/jquery-ui.css" rel="stylesheet" />
<link rel="stylesheet" type="text/css" href="/css/style.css">
<head>
  <script type="text/javascript" src="https://www.google.com/jsapi"></script>
  <script type="text/javascript">
    google.load("visualization", "1", {packages:["controls"]});
    google.load("visualization", "1", {packages:["table"]});
    google.load('visualization', '1', {'packages':['annotationchart']});
```

```
var $symbols_path = "../data/sp500_symbols.txt"
var $dataset_path = "../data/2014-12-03_3yr_sp500_predictions/"
var $entries_per_stock = 7;

var sym_list = [];

$(document).ready(function() {
  // load all stock symbols from file
  var dsv = d3.dsv("\n", "text/plain");
  dsv($symbols_path, function(data){
   sym_list = data.map(function(d) {
    var sym = d["A"];

    // create a placeholder table div and chart div for each symbol
    $( ".container" ).append($("<div class='sym_div' id='" + sym + "'>"
    + "<h4 class='symbol_title'>" + sym + "</h4>" +
    "<div class=chart_div id='" + sym + "'></div>" +
    "<div class=table_div id='" + sym + "'></div></div>"));

    return sym;
   })
  });

  google.setOnLoadCallback(drawAll);
});

// creates tables and charts for each stock
function drawAll() {
 // for each symbol, open it's csv file, and create a new table element
 sym_list.forEach(function(sym) {
   // read in the csv file
   d3.csv($dataset_path + sym + ".csv", function(data) {
    if(data != null && data != undefined) {
      // get relevant columns from dataset as an array of arrays
      var dataset = data.map(function(d) {
          if(d != undefined) {
           return [
             new Date(d["Date"]), +d["Open"],
             +d["Close"], Math.round(+d["Prediction"] * 1000) / 1000,
             $.parseJSON(d["Prediction Correct"].toLowerCase())
            ];
          }else {
            // remove the symbol tables if there is no data for them
            $('#' + sym ).remove();
            return [];
          }
        });

        // add a table and chart for this symbol
        if(dataset != null) {
          drawTable(dataset, sym);
          drawChart(dataset, sym);
        }else {
          $('#' + sym).remove();
        }
      }else {
        $('#' + sym ).remove();
      }
    });
  });

  addSymbolFilter();
}

// set the search bar to filter divs by symbol
function addSymbolFilter() {
   var divs = $('.sym_div');
   $('#search').on('keyup', function() {
      var val = $.trim(this.value);
      divs.hide();
      divs.filter(function() {
          return $(this).attr('id').search(val) >= 0
      }).show();
  });
}

function drawTable(dataset, sym) {
  // initialize table with columns of dataset to use
  var dataTable = new google.visualization.DataTable();
  dataTable.addColumn('date', 'Date');
  dataTable.addColumn('number', 'Open');
  dataTable.addColumn('number', 'Close');
  dataTable.addColumn('number', 'Prediction');
  dataTable.addColumn('boolean', 'Prediction correct?');
  dataTable.allowHtml = true;

  // print the entries for this dataset
  var i;
  var pred_col = 4;

  for(i = 0; i < $entries_per_stock; ++i) {
    row = dataset[i];
    dataTable.addRow(row);

    if(row[pred_col] == true) {
      dataTable.setCell(i, pred_col, true, null,
       {'className': 'green-border center-text bold-green-font large-font'});
    }
  }

  // add the table to the div for this symbol
  var table = new google.visualization.Table(
    document.getElementById(sym).getElementsByClassName("table_div")[0]);
  table.draw(dataTable, {showRowNumber: true});
}

// add a chart for this symbol to show open vs close prices,
   // with annotations that show prediction results
   function drawChart(dataset, sym) {
    var dataTable = new google.visualization.DataTable();
    dataTable.addColumn('date', 'Date');
    dataTable.addColumn('number', 'Open');
    dataTable.addColumn('number', 'Close');
    dataTable.addColumn('string', 'Correct Prediction?');
    dataTable.addColumn('string', 'prediction value');
    // print the entries for this dataset
    var i;
    for(i = 0; i < $entries_per_stock; ++i) {
     row = dataset[i];
     // add a text entry for this row to use for annotation
     // description to show a string with correctness
     // (true/false) and the prediction itself
     row[4] = row[4].toString() + " : " + row[3];
     row.splice(3,1, "Correct Prediction?:");
     dataTable.addRow(row);
    }

    var options = {
     title: 'Opening vs Closing Prices',
     hAxis: {title: 'Date',  titleTextStyle: {color: '#333'}},
     vAxis: {viewWindowMode:'pretty'},
     displayAnnotations: true
    };

    // add the chart to the div for this symbol
    var chart = new google.visualization.AnnotationChart(
     document.getElementById(sym).getElementsByClassName("chart_div")[0]);
    chart.draw(dataTable, options);
   }
  </script>
</head>

<body>
```

```
    <label for="search" class="symbol_title" >Search By Symbol:</label>
    <input type="search" name="filter" id="search" class="input-style-1"/>

    <div class="container">
    </div>
  </body>
</html>
```

References:

1. http://www.acit2k.org/ACIT/2013Proceedings/163.pdf
2. http://www.ccsenet.org/journal/index.php/mas/article/view/4586/3925
3. https://sites.google.com/site/predictingstockmovement/
4. http://www.cs.berkeley.edu/~akar/EE671/report_stock.pdf