Capstone Project

Dave Kessler

Machine Learning Engineer Nanodegree – Udacity

June 24th, 2017

I. Definition

Project Overview

For many years, the movement of stock prices has been studied by financial institutions and individuals wishing to capitalize on repeating patterns. When investing in companies listed on New York Stock Exchange (NYSE), NASDAQ, or other exchanges, the basic strategy is to buy stock shares when the price is low, and sell them later when the price is higher. In so doing, one can gain a profit by obtaining more money than they paid in the first place.

Countless datasets exist that are relevant to stock market mechanics. Many of these datasets are freely available to the public, while others can be obtained for a fee. This project uses historical stock price data provided by Yahoo! Finance for free to the public.

In this project, I created a single Python 2.7 script that is intended to be converted into a standalone executable program for users not familiar with Python. This program is capable of importing historical stock price data, processing and displaying it, and finally using Machine Learning to make future stock price predictions. With this kind of program, an individual or financial institution could make better stock trading or investment decisions. Using a program like this myself was my main motivation for solving this kind of problem.

Problem Statement

The problem to be solved in this project is to predict future stock prices. Since prices are continuous values, this is a regression task. The solution was achieved using a Machine Learning algorithm called a Support Vector Machine (SVM). The stocks of four companies are considered during this project, but the solution could be applicable to a wider variety of stocks.

This solution comes in the form of a Python 2.7 script. It creates a Graphical User Interface (GUI) for the user to interact with easily. This solution is intended to later be converted into a standalone executable (.exe) file for people who are not familiar with Python or the other packages used in this Machine Learning model.

Metrics

Percent error will be the metric used to evaluate the performance of the model. It will compare the predicted adjusted closing price produced by the model to the actual adjusted closing price supplied by the historical stock data, and represented as a percentage.

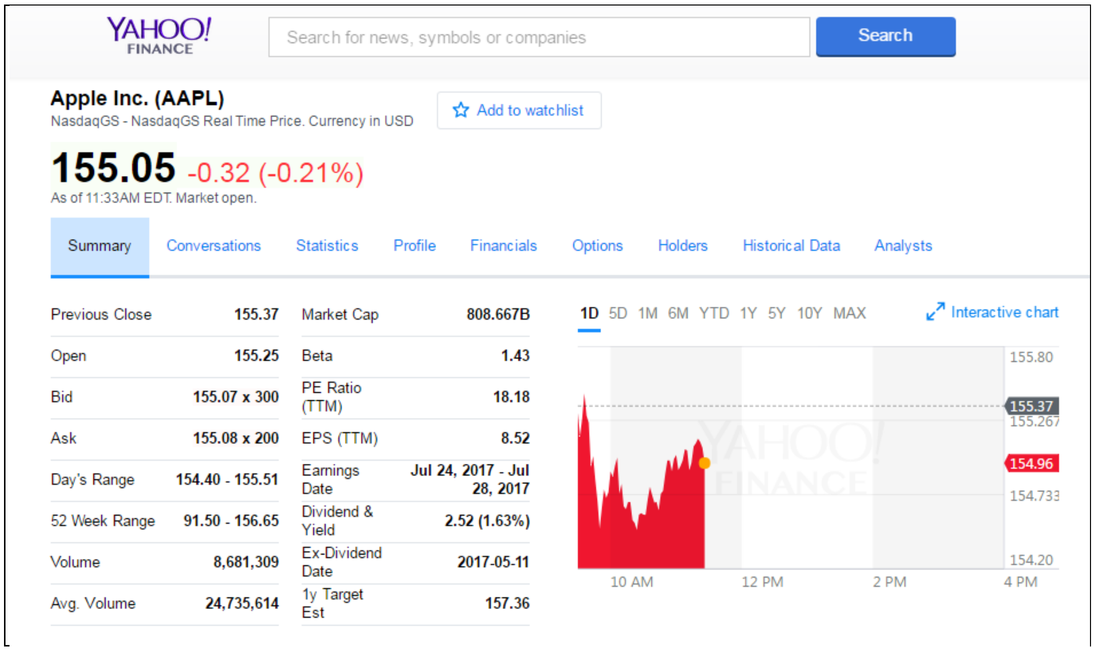
$$\%\ Error = \left(\frac{Predicted\ Adjusted\ Close\ Price}{Actual\ Adjusted\ Close\ Price} - 1\right) * 100$$

There are other regression performance metrics such as Mean Absolute Error, Mean Squared Error, and R2 Score built into many Machine Learning libraries. These are all great for explaining how close a predicted value is to the actual value (Brownlee). However, this application is designed for users who are perhaps not familiar with Machine Learning or its associate metrics. Percent Error is a performance metric that is more widely understood.

II. Analysis

Data Exploration

Yahoo! Finance gives free access to millions of data points related to stocks. There are plenty of other sources of stock market data, but this one was chosen because it is easy for users to navigate, research, and use. The public can start by visiting https://finance.yahoo.com/, and searching for a company or stock symbol. They can then view many things about that particular company such as; its current stock price and price change from the previous trading day, stock price statistics, company financial statistics, historical stock price data, and more.



Apple Inc. (AAPL)
NasdaqGS - NasdaqGS Real Time Price. Currency in USD

155.05 -0.32 (-0.21%)
As of 11:33AM EDT. Market open.

Previous Close	155.37	Market Cap	808.667B
Open	155.25	Beta	1.43
Bid	155.07 x 300	PE Ratio (TTM)	18.18
Ask	155.08 x 200	EPS (TTM)	8.52
Day's Range	154.40 - 155.51	Earnings Date	Jul 24, 2017 - Jul 28, 2017
52 Week Range	91.50 - 156.65	Dividend & Yield	2.52 (1.63%)
Volume	8,681,309	Ex-Dividend Date	2017-05-11
Avg. Volume	24,735,614	1y Target Est	157.36

Figure 1 Yahoo! Finance website displaying Apple Inc. (AAPL) stock data

Historical stock price data downloaded from this site and used for this project comes in the form of a Comma Separated Variable (.csv) file. The data labels are 'Date', 'Open', 'High', 'Low', 'Close', 'Adj Close', and 'Volume'. The frequency of the data is one day increments, and only for trading days where the stock market was open.

- 'Date' (string) - Dates displayed in mm/dd/yyyy format
- 'Open' (float) – Price value recorded at the open (9:30am EST) of trading
- 'High' (float) – Highest price recorded over the entire day
- 'Low' (float) – Lowest price recorded
- 'Close' (float) – Price recorded at the end (4:00pm EST) of trading
- 'Adj Close' (float) – Closing price adjusted to include any distributions and corporate actions that occurred before the next day's open ("Adjusted Closing Price")
- 'Volume' (integer) - Number of shares traded that day

Table 1 Sample of historical stock price data for Apple, Inc. (AAPL)

Date	Open	High	Low	Close	Adj Close	Volume
1/2/2013	72.09232	72.24593	70.5055	549.03	71.46877	1.4E+08
1/3/2013	71.31908	71.55209	70.42349	542.1	70.56668	88241300
1/4/2013	69.8989	70.11498	68.44875	527	68.60107	1.49E+08
1/7/2013	67.9502	68.90047	67.06503	523.9	68.19753	1.21E+08
1/8/2013	68.88876	69.23764	67.85258	525.31	68.38107	1.15E+08

The stocks of four companies were analyzed during this project: Alphabet Inc. (GOOG), Amazon.com, Inc. (AMZN), Microsoft Corporation (MSFT), and Apple Inc. (AAPL). Basic statistics of their adjusted closing prices and volume during the period from 1/2/2013 – 12/30/2016 are shown below.

Table 2 'GOOG', 'AMZN', 'MSFT', and 'AAPL' 1/2/2013 – 12/30/2016

	GOOG		AMZN		MSFT		AAPL	
	Adj. Close	Volume	Adj. Close	Volume	Adj. Close	Volume	Adj. Close	Volume
Min	350.12	7900	248.23	1091200	23.41	8409600	51.13	11475900
Max	813.11	23219400	844.36	23856100	62.88	248428500	127.44	365213100
Average	586.44	2644698	452.06	3742649	41.7	37228266	91.96	63753553
STD	120.3	1760516	175.55	2242331	9.76	20706570	21.77	37991293

As can be seen from this table, there is a considerable difference between all of the minimum and maximum values. This is due to gradual price increase or decrease over time as well as the variation in trading excitement over different days. The considerable spread in values makes these four stocks challenging for any prediction algorithm, and this is why they were chosen.

The nature of stock prices prevents the possibility of any kind of price outlier. The reasons for this deal with stock market mechanics, and are beyond the scope of this project. Volume can spike drastically some days, and this is often accompanied by some kind of price change. Because of this, no data was excluded due to exceptionally high or low volumes.

Exploratory Visualization

A relevant characteristic of this data is the daily adjusted closing price of the stock. This feature is often used when examining price trends or performing a detailed analysis on historical returns. It is very useful because it represents a company's equity beyond a simple market price ("Adjusted Closing Price").

Another relevant characteristic is the volume of shares traded each day. If the price of a stock changes beyond its normal daily fluctuation, volume can be used to measure the strength of that movement. The higher the volume during the price movement, the more powerful it is ("Volume").

Below are plots of both adjusted closing price and volume for the four stocks analyzed in this project: GOOG, AMZN, MSFT, and AAPL for the dates of 1/2/2013 – 12/30/2016. These kinds of plots are common in the financial industry when displaying stock price data.

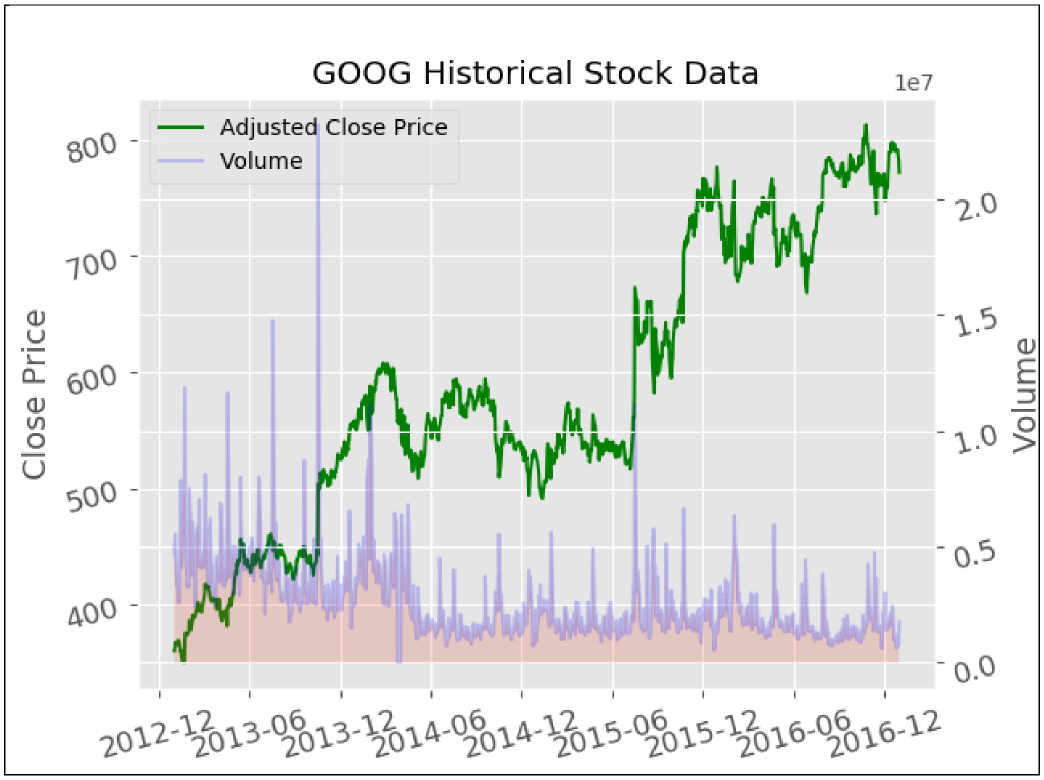


Figure 2 Adjusted closing price and volume for Alphabet Inc. (GOOG) 1/2/2013-12/30/2016

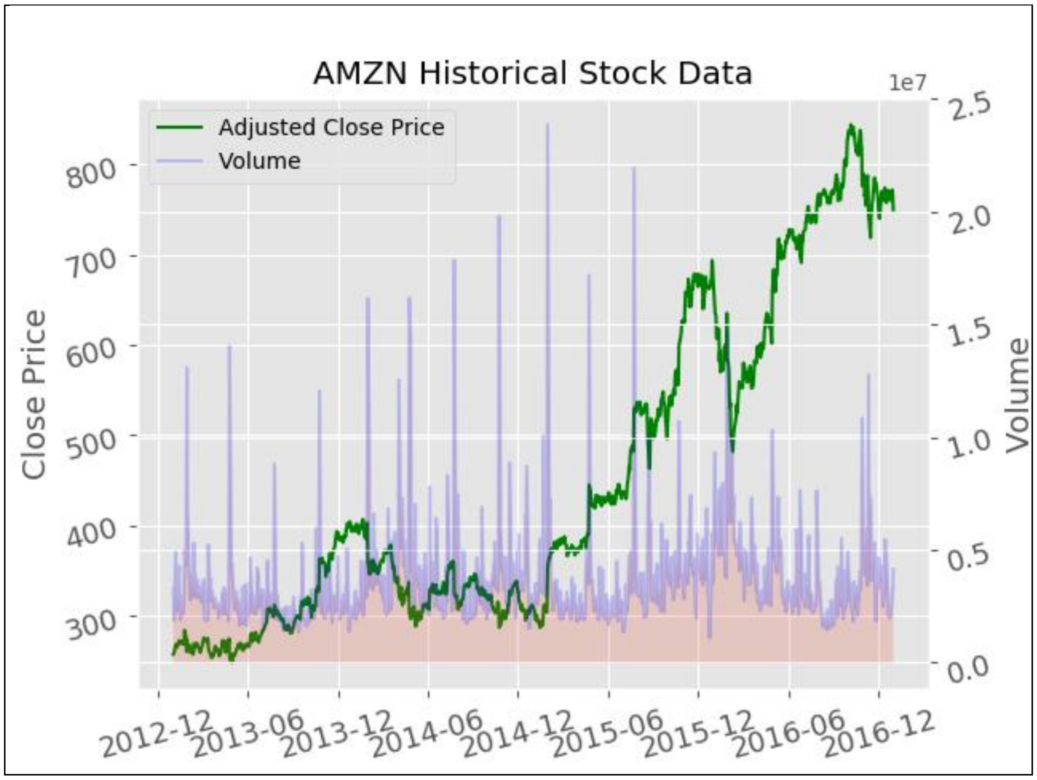


Figure 3 Adjusted closing price and volume for Amazon.com, Inc. (AMZN) 1/2/2013-12/30/2016

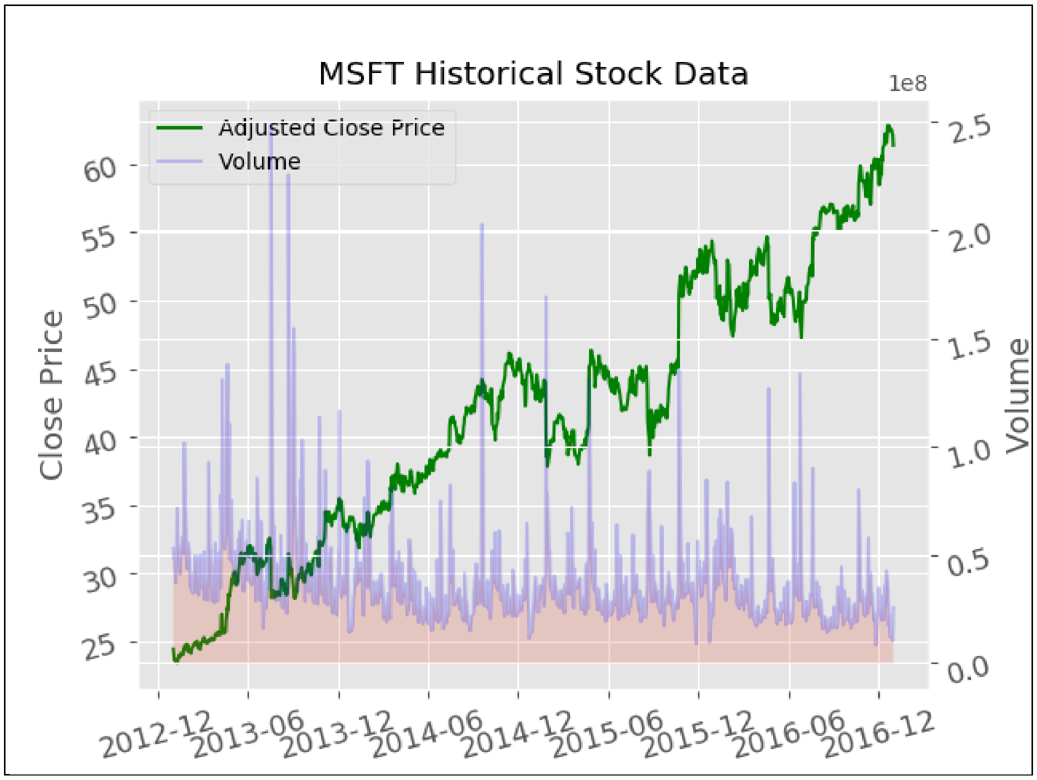


Figure 4 Adjusted closing price and volume for Microsoft, Inc. (MSFT) 1/2/2013-12/30/2016

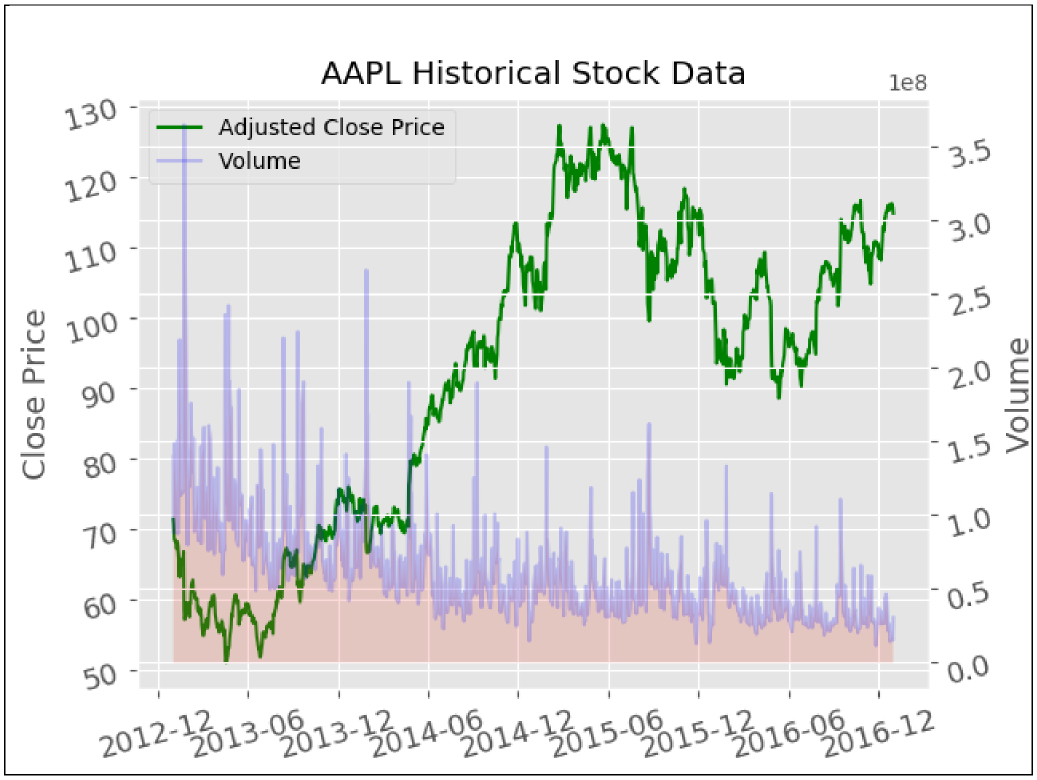


Figure 5 Adjusted closing price and volume for Apple, Inc. (AAPL) 1/2/2013-12/30/2016

Algorithms and Techniques

For this project, I chose three different Machine Learning models to compare against each other. Predicting future stock prices is by its nature a regression problem, as the prices are continuous values. The three regression models that were chosen are Support Vector Machine (SVM), Random Forest (RF), and K-Nearest Neighbors (KNN).

Support Vector Machines (SVMs) work by finding a *maximum margin hyperplane* which maximizes separation between data classes. The training samples that are closest to this hyperplane are called the *support vectors* (Kim). Support Vector Regression builds upon this and use kernels to specify a *margin of tolerance* (epsilon) in order to provide real number prediction output ("Support Vector Regression").

Random Forest (RF) regressors are estimators that fit a number of classifying decision trees on various subsamples of the dataset. Then they use averaging to improve the prediction accuracy and control overfitting to the data ("RandomForestRegressor"). The scikit-learn version of this model combines classifiers by averaging their probabilistic prediction, instead of letting each classifier vote for a single class ("1.11. Ensemble Methods").

K-Nearest Neighbors (KNN) models work by assigning a label to a requested point as being the average of the ***k***-nearest point labels to it. The basic regression model uses uniform weights in that each point in the local neighborhood contributes uniformly to the classification of a query point ("1.6. Nearest Neighbors").

The un-tuned versions of these models will use the default parameters provided in the scikit-learn packages. Parameter tuning was then used to better the model's accuracy. Scikit-learn's GridSearchCV module performed the tuning by taking a dictionary of parameters and their values, and testing all of their possible combinations. The blend that produced the best results on the training data was passed to final model for future price prediction.

All three of these models work well with stock data because they are regression models. They can learn complex relationships between the features and labels, and produce the required continuous output predictions. The historical stock data used by these models are composed of only continuous values, and so the data can be used directly by the models. No transformation from categorical to continuous data is needed. The performance of each model is discussed later in this project.

Benchmark

The benchmark for this project will be a Simple Moving Average (SMA). This basic model is often used by traders to help identify stock price trends. It calculates the adjusted closing price for a stock on a certain day as being the average of the adjusted closing prices for that stock over the previous (5, 10, 20, etc.) trading days ("Simple Moving Average - Technical Analysis").

The window used for testing the model produced by this project is four trading days. This should be an adequate amount of time for the price of the stock to considerably change, providing good insight into the performance of the Machine Learning model. SMAs are very basic, and so it will be the perfect model to compare against.

Table 3 Example: Simple Moving Average (SMA) for Google (GOOG) with a 4-day window

Date	Adj. Close	SMA(4)
11/29/2016	770.84	N/A
11/30/2016	758.04	N/A
12/1/2016	747.92	N/A
12/2/2016	750.5	756.83

For this project, both the model and benchmark were tasked with making adjusted closing price predictions for 12/09/2016. This date is four trading days into the future from 12/05/2016. This date was selected because it was near the end of the datasets used in this project. Using this date ensured that the majority of the datasets (all dates preceding and including 12/05/2016) were used for training. The benchmark's predictions and the corresponding percent errors are shown below for the four stocks analyzed in this project.

Table 4 Benchmark predictions for 12/09/2016, four days into future from 12/05/2016

	Actual Adj. Close	SMA(4)	Bench Pct. Err.
GOOG	789.29	767.31	2.78
AMZN	768.66	765.46	0.42
MSFT	61.25	59.93	2.16
AAPL	112.99	109.63	2.98

III. Methodology

Data Preprocessing

The first step in preprocessing is fetching the data for the required stock for the required dates. This data is downloaded by the user from the Yahoo! Finance website and stored as CSV files in the "Daily_SP_Downloads" folder on the user's computer. Data is imported by the program into a Pandas dataframe.

Historical stock price plotting features offered by the program will allow the user to see the adjusted closing prices of the stock they requested over the dates they requested. The user will then select the date that will be the last training date, and the dates for which they would like the model to predict the adjusted closing price.

Once the program receives the command from the user to predict the future prices, it calculates the ***n*** number of days from the last training date to the requested future date. An example calculation where ***n*** = 3 is shown below:

- User requests Last Training Date = 12/5/2016
- User requests First Predicted Date = 12/8/2016

Table 5 Sample historical stock price data for Google (GOOG)

Date	Open	High	Low	Close	Adj Close	Volume
12/1/2016	757.44	759.85	737.025	747.92	747.92	3017900
12/2/2016	744.59	754	743.1	750.5	750.5	1452500
12/5/2016	757.71	763.9	752.9	762.52	762.52	1394200
12/6/2016	764.73	768.83	757.34	759.11	759.11	1690700
12/7/2016	761	771.36	755.8	771.19	771.19	1761000
12/8/2016	772.48	778.18	767.23	776.42	776.42	1488100

The program then makes a copy of the dataframe containing the stock data and adds new columns to this dataframe. Some of these columns will be some of the features and labels used later by the model. The other added columns will hold values for the Simple Moving Average (SMA) benchmark. The new columns are:

- '{***n***} Return' : (Label)

$$\frac{Adjusted\ Closing\ Price\ on\ Requested\ Future\ Date}{Adjusted\ Closing\ Price\ on\ Last\ Training\ Date}$$

- 'Roll {***n***} ADV' : (Feature) Rolling Average Daily Volume -

$$\frac{Volume\ Traded\ Over\ \mathbf{n}\ Trading\ Days\ Ending\ on\ Last\ Training\ Date}{\mathbf{n}\ Number\ of\ Trading\ Days\ Ending\ on\ Last\ Training\ Date}$$

- ''V Div {***n***} ADV' : (Feature) Volume Divided by Rolling Average Daily Volume -

$$\frac{Volume\ of\ Shares\ Traded\ on\ Last\ Training\ Date}{Roll\ \{\mathbf{n}\}ADV}$$

- 'SMA {***n***}' : (Feature) Simple Moving Average -

$$\frac{Adjusted\ Closing\ Price\ for\ Each\ of\ \mathbf{n}\ Trading\ Days\ Ending\ on\ Last\ Training\ Date}{\mathbf{n}\ Number\ of\ Trading\ Days\ Ending\ on\ Last\ Training\ Date}$$

- 'Roll {***n***} STD' : (Feature) Rolling Standard Deviation – *Standard Deviation of the Adjusted Closing Price over the* **n** *number of trading days ending on the last training date*

- 'Bench_PAC' : *Simple Moving Average Benchmark's Predicted Adjusted Closing Price for the future date requested*

- 'Bench_Pct_Err': Benchmark's Percent Error-
$$\left(\frac{Bench\_PAC}{Adjusted\ Closing\ Price\ for\ the\ Future\ Date\ Requested} - 1\right) * 100$$

Next, all of the features to be used by the model are gathered into a new dataframe *X_all*, and their corresponding labels are gathered together into another new dataframe *y_all*.

X_all and y_all are then split into training and testing sets to produce X_train, X_test, y_train, and y_test. This is done by using the last training date entered by the user to be the last entry in the X_train and y_train dataframes. Everything that follows this date goes into X_test and y_test dataframes.

Splitting is done this way to avoid 'look-ahead bias'. This kind of bias is created by the use of data in a simulation that would not have been known during the period being analyzed. This will usually lead to inaccurate results in the simulation ("Look-Ahead Bias").

Yahoo! Finance structures its historical stock price information in a very consistent way, and I have never seen any missing data for any of the stocks or dates that I have requested. This is very helpful, as it makes preprocessing for this program a lot easier. That being said, there are certain cases in which the other preprocessing steps discussed above may produce Null values for the earliest date entries in the corresponding dataframes.

In order to account for this, the program applies backfilling to the dataframes to fill any missing values with the next available value. Then it applies forward filling to fill any values missed by the backfilling.

Implementation

The end result of this project is designed to be used by the lay-person. The Python script could be converted to a standalone executable (.exe) file that contains all of the necessary Python, Python packages, and the code included in this project. This conversion could be done using a service like Py2exe (http://www.py2exe.org/). By having the program be a single .exe file, the user would not be required to have Python or any of its packages installed, and will not need any programming knowhow or experience.

NOTE: For the sake of this submission to Udacity for review, only the original Python script will be submitted, with a relevant Readme document.

The Graphical User Interface (GUI) was designed with the help of QT Designer (https://www.qt.io/). This program helped to create the windows and data fields. It also helped to size and place the buttons, along with any added tooltips that could help the user understand functionality.

The GUI is designed to automatically create a folder on the user's computer in the same directory where the Python script exists. This folder is to hold the .csv files that the program automatically downloads from the Yahoo! Finance system using the pandas_datareader Python package. However, changes to the Yahoo! Finance system in May-June 2017 have made it impossible for the package to download data properly. Once an updated version of this package is released, the user will be able to interact with only the GUI and accomplish all plotting and price prediction tasks. **NOTE: For now, the user will have to download all stock data manually, as discussed above.**

When the user first runs the program, a window appears into which they can enter stock symbol and date information. There are default date values already shown in the window. The user may either type new dates, or chose a date from the dropdown calendar built into the window.

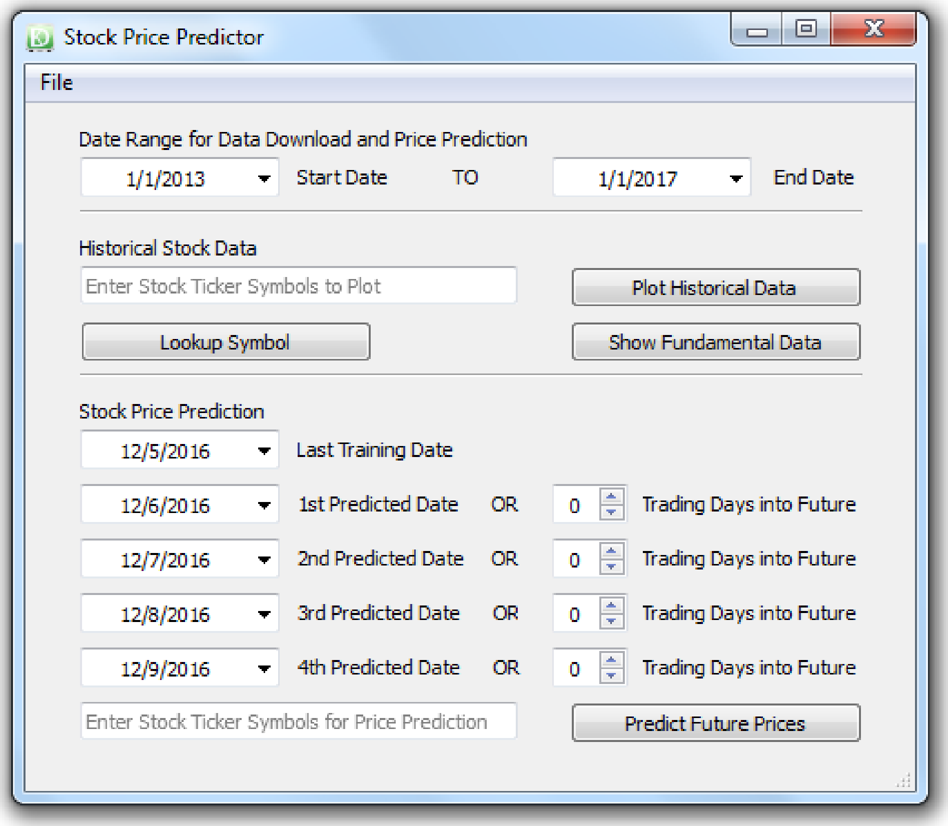


Figure 6 Stock price predictor start window

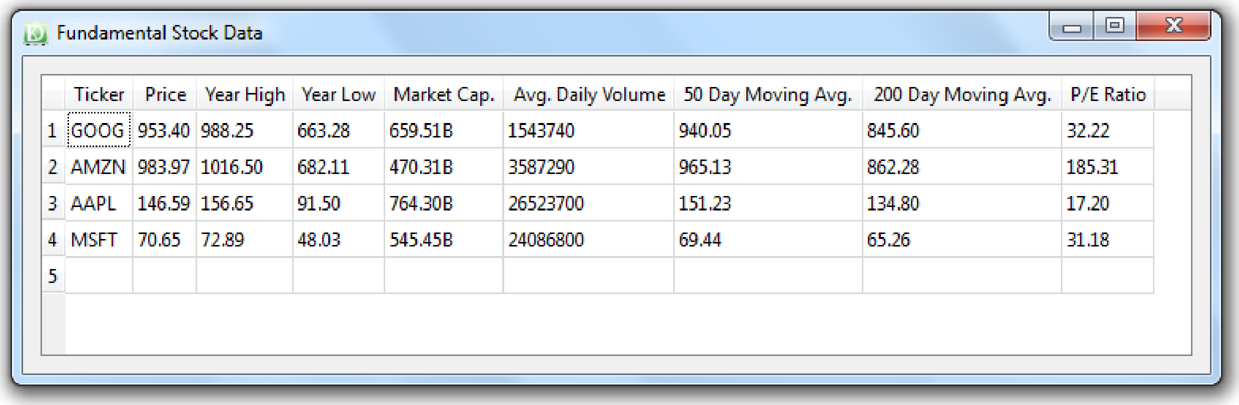
Many of the buttons and fields on this start window have tooltips that will appear when the user hovers over them. The tooltips are as follows:

- **Start Date -** The beginning of the range of dates to be used for data download and/or stock price prediction.
- **End Date -** The end of the range of dates to be used for data download and/or stock price prediction.
- **Historical Stock Data -** (MAX 5) Enter stock ticker symbols to plot, separated by commas.
- **Lookup Symbol -** Opens web page to help find a stock's ticker symbol.

- **Plot Historical Data -** Plots the daily historical stock price and volume information for a given stock symbol.
- **Show Fundamental Data -** Plots the fundamental data for a given stock symbol.
- **Last Training Date -** Last day used for training the price prediction model.
- **1st Predicted Date -** First day for which the closing price will be predicted. Must come after the Last Training Date.
- **2nd Predicted Date -** (Optional) Second day for which the closing price will be predicted. Must come after the Last Training Date.
- **3rd Predicted Date -** (Optional) Third day for which the closing price will be predicted. Must come after the Last Training Date.
- **4th Predicted Date -** (Optional) Fourth day for which the closing price will be predicted. Must come after the Last Training Date.
- **Predict Future Prices -** (MAX 5) Enter stock ticker symbols for price prediction, separated by commas.

'Plot Historical Data' produces a chart like the one shown above as 'Figure 7 Adjusted closing price and volume for Apple, Inc. (AAPL) 1/2/2013-12/30/2016'

'Show Fundamental Data' produces a table containing financial data for up to five stock symbols entered by the user. The program retrieves this data from the Yahoo! Finance system, and does not rely on data saved manually by the user.


Fundamental Stock Data

	Ticker	Price	Year High	Year Low	Market Cap.	Avg. Daily Volume	50 Day Moving Avg.	200 Day Moving Avg.	P/E Ratio
1	GOOG	953.40	988.25	663.28	659.51B	1543740	940.05	845.60	32.22
2	AMZN	983.97	1016.50	682.11	470.31B	3587290	965.13	862.28	185.31
3	AAPL	146.59	156.65	91.50	764.30B	26523700	151.23	134.80	17.20
4	MSFT	70.65	72.89	48.03	545.45B	24086800	69.44	65.26	31.18
5									

Figure 8 Fundamental stock data retrieved from Yahoo! Finance

The implementation of all three models chosen for this project was conducted by using the Machine Learning libraries provided by scikit-learn. The un-tuned models use the defaults values for the parameters that are built into the models. The documentation for these models, as well as examples can be found at http://scikit-learn.org/stable/. The models all take the exact same data in the same structure that is produced by the preprocessing steps.

Implementing the three models was completed through the following lines of code:

- Importing required libraries
```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from sklearn.ensemble import RandomForestRegressor
```

- Building classifier – only one used at a time, others must be commented
```
self.clf = KNeighborsRegressor()
self.clf = SVR(cache_size = 600)
self.clf = RandomForestRegressor(random_state = 5)
```

- Fitting classifier to training data (un-tuned models only)
```
self.clf.fit(self.X_train, self.y_train.values.ravel())
```

- Importing libraries for GridSearchCV (tuned models only)
```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import r2_score
```

- Declaring parameters for GridSearchCV (tuned models only)
- Only one block used at a time, others must be commented
```
#These parameters are for a random forest regressor
self.parameters = {'n_estimators' : [2, 3, 4, 5, 16],
                     'min_samples_split' : [2, 3, 4, 5]}

#These parameters are for a SVR
self.parameters = {'C' : [0.05, 0.08, 0.1, 0.5, 0.9, 1],
                  'gamma' : [ 0.0001, 0.001, 0.01]}

#These parameters are for a KNeighborsRegressor
self.parameters = {'n_neighbors' : [3, 5, 7, 9],
                  'leaf_size' : [10, 20, 30, 40],#, 'sigmoid'],
                  'weights' : ['uniform', 'distance']}
```

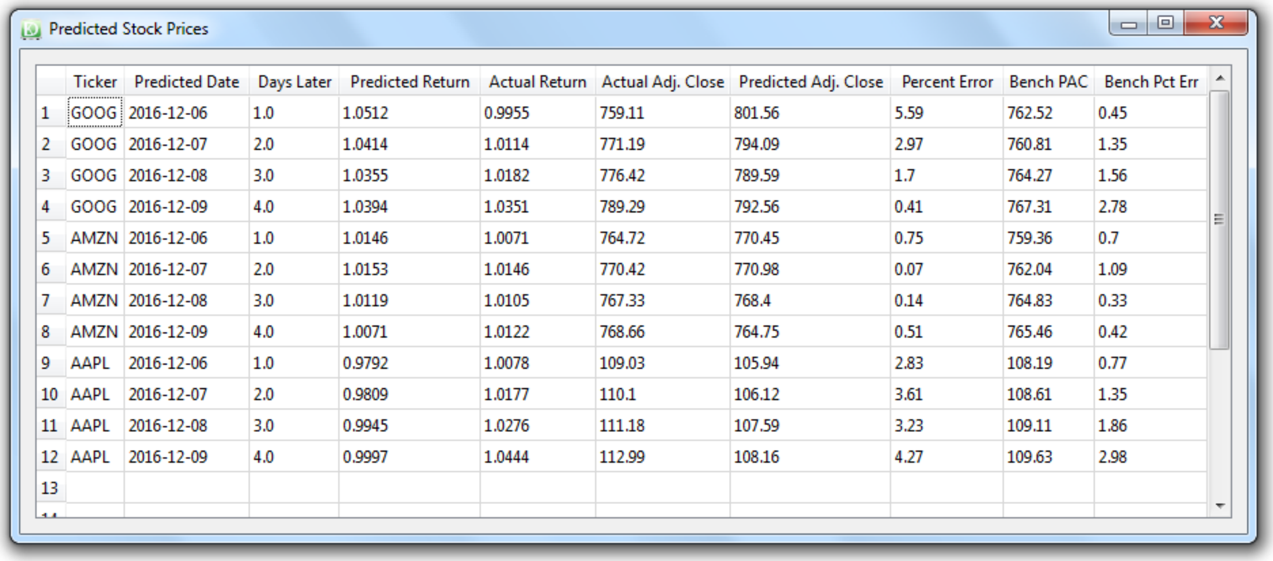
- Making an r2_score scoring object for use with GridSearchCV
```
self.scorer = make_scorer(r2_score, greater_is_better = False)
```

- Performing grid search on the classifier using 'scorer' as the scoring method
```
self.grid_obj = GridSearchCV(self.clf, self.parameters, scoring =
            self.scorer, n_jobs=-1)
```

- Fitting the grid search object to the training data and finding the optimal parameters
  ```
  self.grid_fit = self.grid_obj.fit(self.X_train,
                  self.y_train.values.ravel())
  ```
- Getting the estimator
  ```
  self.best_clf = self.grid_fit.best_estimator_
  self.clf = self.best_clf
  ```

The benchmark used to compare against the final chosen model is a Simple Moving Average (SMA) with a four-day window. The adjusted closing price predicted by the SMA is calculated during the preprocessing stage, and later displayed along with the results produced by the model.

The metric used for this project is the difference between the predicted adjusted closing price and the actual closing price, represented as a percentage error. The scores of the model and the benchmark are calculated before being gathered with other pertinent information and displayed to the user in a results window when the user clicks 'Predict Future Prices'.



Predicted Stock Prices

	Ticker	Predicted Date	Days Later	Predicted Return	Actual Return	Actual Adj. Close	Predicted Adj. Close	Percent Error	Bench PAC	Bench Pct Err
1	GOOG	2016-12-06	1.0	1.0512	0.9955	759.11	801.56	5.59	762.52	0.45
2	GOOG	2016-12-07	2.0	1.0414	1.0114	771.19	794.09	2.97	760.81	1.35
3	GOOG	2016-12-08	3.0	1.0355	1.0182	776.42	789.59	1.7	764.27	1.56
4	GOOG	2016-12-09	4.0	1.0394	1.0351	789.29	792.56	0.41	767.31	2.78
5	AMZN	2016-12-06	1.0	1.0146	1.0071	764.72	770.45	0.75	759.36	0.7
6	AMZN	2016-12-07	2.0	1.0153	1.0146	770.42	770.98	0.07	762.04	1.09
7	AMZN	2016-12-08	3.0	1.0119	1.0105	767.33	768.4	0.14	764.83	0.33
8	AMZN	2016-12-09	4.0	1.0071	1.0122	768.66	764.75	0.51	765.46	0.42
9	AAPL	2016-12-06	1.0	0.9792	1.0078	109.03	105.94	2.83	108.19	0.77
10	AAPL	2016-12-07	2.0	0.9809	1.0177	110.1	106.12	3.61	108.61	1.35
11	AAPL	2016-12-08	3.0	0.9945	1.0276	111.18	107.59	3.23	109.11	1.86
12	AAPL	2016-12-09	4.0	0.9997	1.0444	112.99	108.16	4.27	109.63	2.98
13										

Figure 9 Screenshot of stock price predictor results window

Refinement

The three Machine Learning models that were tested during this project were all used without any tuned parameters at first. The parameters were set to be their default values supplied in the scikit-learn libraries. The results of the un-tuned models are shown below.

All of the models were then tuned using scikit-learn’s GridSearchCV. This is an approach to parameter tuning that builds and evaluates models for each combination of algorithm parameters specified in a grid (Brownlee).

The parameters that were tuned for the Support Vector Machine (SVM) were ‘C’ and ‘gamma’, as the base kernel used is the Radial Basis Function (RBF). ‘C’ is the penalty parameter of the error term, and trades off misclassification of training samples against simplicity of the decision surface. It has a default value of 1, but if the data is noisy at all, it is important to reduce the value of ‘C’. Stock price data by its very nature is noisy, and so values of ‘C’ ranging from 0.05 to 1 were tried in this project. ‘Gamma’ defines how much influence a single training sample has. The larger ‘gamma’ is, the closer other examples must be in order to be affected (1.4. Support Vector Machines).

The parameters tuned for the Random Forest (RF) regressor model were ‘n_estimators’ and ‘min_samples_split’. ‘N_estimators’ controls the number of trees that were used in fitting the model to the data. Values of 2 - 16 were used in the GridSearchCV parameter grid. ‘Min_samples_split’ is the number of samples required to split an internal node. When the model is fitting to the data, there must be at least this many samples in a particular area for the decision boundary to split, adding definition and complexity in that area. Values ranging from 2 - 5 were tried in this project (“RandomForestRegressor”).

For the K-Nearest Neighbors (KNN) regressor, I tuned three different parameters: ‘n_neighbors’, ‘leaf_size’, and ‘weights’. The number of neighbors to be used for queries is reflected in the ‘n_neighbors’ parameter. Values of 3 -9 were tried in the model. The ‘leaf_size’ parameter reflects the number of samples at which to switch to brute-force. This parameter does not affect the results of a query, but can significantly affect the speed of a query and the amount of memory required to store the constructed tree. The amount of memory needed scales as approximately the number of samples divided by the ‘leaf_size’ (“KDTree”). Leaf sizes of 10 to 40 were tested in the model. ‘Weights’ lists the weighting algorithms to be used by the model. ‘Uniform’ weighting gives all points in a neighborhood equal weighting. ‘Distance’ weighting gives neighbors that are closer to a point greater weighting than neighbors that are farther away (“KNeighborsRegressor”).

Below, the accuracy of predictions of the three models discussed above are compared. The adjusted closing price of four different stocks (‘GOOG’, ‘AMZN’, ‘MSFT’, and ‘AAPL’) was predicted five trading days after the last training date. The first table shows the un-tuned results, and the tuned results are shown in the second table.

Table 6 Un-tuned prediction results

Ticker	Predicted Date	Days Later	Percent Error - KNN	Percent Error - SVM	Percent Error - RF	Bench Pct Err
GOOG	12/12/2016	5	2.34	0.53	6.14	2.23
AMZN	12/13/2016	5	4.04	0.59	0.67	0.79
MSFT	12/14/2016	5	5.7	2.67	4.23	2.03
AAPL	12/15/2016	5	5.56	3.95	4.69	1.83
				Best ML model		Benchmark

Table 7 Tuned prediction results

Ticker	Predicted Date	Days Later	Percent Error - KNN	Percent Error - SVM	Percent Error - RF	Bench Pct Err
GOOG	12/12/2016	5	2.19	0.53	4.65	2.23
AMZN	12/13/2016	5	1.38	0.59	0.84	0.79
MSFT	12/14/2016	5	3.82	2.67	4.31	2.03
AAPL	12/15/2016	5	6.1	3.95	4.85	1.83
				Best ML model		Benchmark

As can be seen from these results, tuning made the biggest improvement for the KNN model. The results of the SVM remained unchanged and other parameters or preprocessing methods may need to be implemented to produce improvements during GridSearchCV tuning. The tuning for the RF model produced slightly worse results compared to the un-tuned model, which tells me that further parameter tuning may need to occur to produce improvements.

IV. RESULTS

Model Evaluation and Validation

The final model chosen based upon the previous trials is the Support Vector Machine (SVM). The reasoning for this is simply that it produced the most accurate price prediction for four different stocks five days after the last training date, as shown above.

The parameters for the SVM are tuned every time the model is run, and the best combination is used to make each individual prediction requested by the user. Since the user may be able to enter multiple dates for prediction, the optimum combination may differ from request to request. However, all of the queries are subject to the following ranges of parameters:

- ‘C’ – [0.05, 0.08, 0.1, 0.5, 0.9, 1]
- ‘Epsilon’ = 0.1 (default)
- ‘Kernel’ = ‘rbf’ (default)
- ‘Degree’ = 3 (default)
- ‘Gamma’ - [0.0001, 0.001, 0.01]
- ‘Coef0’ = 0.0 (default)
- ‘Shrinking’ = True (default)
- ‘Tol’ = 1e^-3 (default)
- ‘Cache_size’ = 600
- ‘Verbose’ = False (default)
- ‘Max_iter’ = -1 (default)

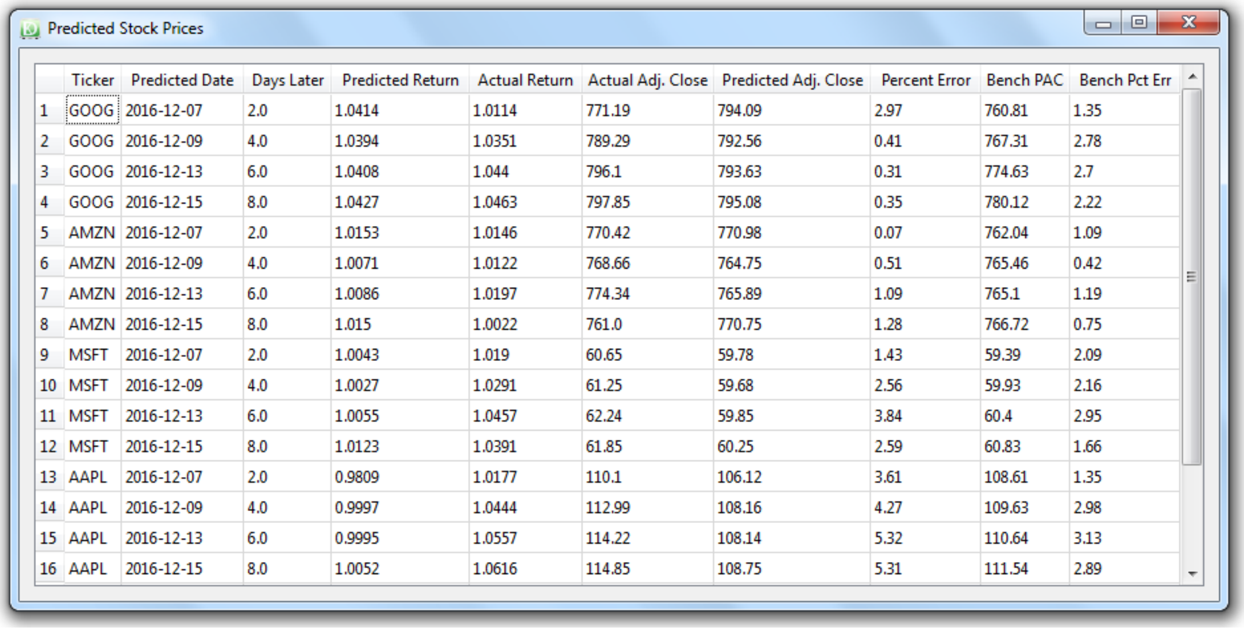
This model was tested against various stocks over varying query dates. Some companies traded on the stock markets are extremely large, having a worth of billions of dollars. Others are only worth several million. Because of this, their stocks have different values, and the price trends of those stocks fluctuate differently over time. The model used in this project tunes parameters before every prediction, helping the model better generalize to unseen data.

The results provided by the final model can be trusted in that any of the prediction dates’ values going into the model can be verified easily. The user has .csv files for the stocks and dates that they are interested in, and the stock price predictor’s results window shows some pertinent information about the requested dates/stocks. If the user likes, they can compare the information in the two locations to make sure that everything matches up correctly.

Justification

In order to see how the model holds up against different inputs, four different stocks were queried for dates that were 2, 4, 6, and 8 days after the last training date. This test composed 16 individual tests.

Table 8 Tuned Support Vector Machine (SVM), 4 stocks * 4 dates



	Ticker	Predicted Date	Days Later	Predicted Return	Actual Return	Actual Adj. Close	Predicted Adj. Close	Percent Error	Bench PAC	Bench Pct Err
1	GOOG	2016-12-07	2.0	1.0414	1.0114	771.19	794.09	2.97	760.81	1.35
2	GOOG	2016-12-09	4.0	1.0394	1.0351	789.29	792.56	0.41	767.31	2.78
3	GOOG	2016-12-13	6.0	1.0408	1.044	796.1	793.63	0.31	774.63	2.7
4	GOOG	2016-12-15	8.0	1.0427	1.0463	797.85	795.08	0.35	780.12	2.22
5	AMZN	2016-12-07	2.0	1.0153	1.0146	770.42	770.98	0.07	762.04	1.09
6	AMZN	2016-12-09	4.0	1.0071	1.0122	768.66	764.75	0.51	765.46	0.42
7	AMZN	2016-12-13	6.0	1.0086	1.0197	774.34	765.89	1.09	765.1	1.19
8	AMZN	2016-12-15	8.0	1.015	1.0022	761.0	770.75	1.28	766.72	0.75
9	MSFT	2016-12-07	2.0	1.0043	1.019	60.65	59.78	1.43	59.39	2.09
10	MSFT	2016-12-09	4.0	1.0027	1.0291	61.25	59.68	2.56	59.93	2.16
11	MSFT	2016-12-13	6.0	1.0055	1.0457	62.24	59.85	3.84	60.4	2.95
12	MSFT	2016-12-15	8.0	1.0123	1.0391	61.85	60.25	2.59	60.83	1.66
13	AAPL	2016-12-07	2.0	0.9809	1.0177	110.1	106.12	3.61	108.61	1.35
14	AAPL	2016-12-09	4.0	0.9997	1.0444	112.99	108.16	4.27	109.63	2.98
15	AAPL	2016-12-13	6.0	0.9995	1.0557	114.22	108.14	5.32	110.64	3.13
16	AAPL	2016-12-15	8.0	1.0052	1.0616	114.85	108.75	5.31	111.54	2.89

The average percent error of the model's predictions in each of the 16 tests is 2.25, while the average percent error produced by the benchmark is 1.98. This shows that the benchmark is overall more accurate than the SVM model.

However, it should be noted that for 'GOOG' predictions, the SVM had significantly lower error than the benchmark model. The SVM performed slightly better than the benchmark when making predictions for 'AMZN'. The benchmark's predictions for 'MSFT' were slightly better than the SVM's, and its predictions for 'AAPL' were significantly better.

This tells me that the SVM model could be improved by examining these results deeper. Perhaps there were higher volumes of a company's stock being traded during this period in time. Maybe the company merged with another and the price jumped up or down because of that news. These kinds of correlations go deeper into financial theory, and are beyond the scope of this project.

The benchmark posed earlier in this project is the Simple Moving Average over a four-day window. Looking at the model's predictions, we see that the average error is 1.94. The average error of the benchmark is 2.09. Because of this, the goal of the project is satisfied.

V. Conclusion

Free-Form Visualization

A feature that could be more closely examined is the relationship between the volume of stocks traded during a period of time and the change in adjusted closing price during that same period. Below is a plot of historical stock prices for 'AMZN'. Enclosed in red ovals are periods where there was a spike in traded stock volume and the price of the stock decreased. The green ovals enclose periods where there was a volume spike and the price of the stock increased. This correlation could be studied further to produce new features for the model.

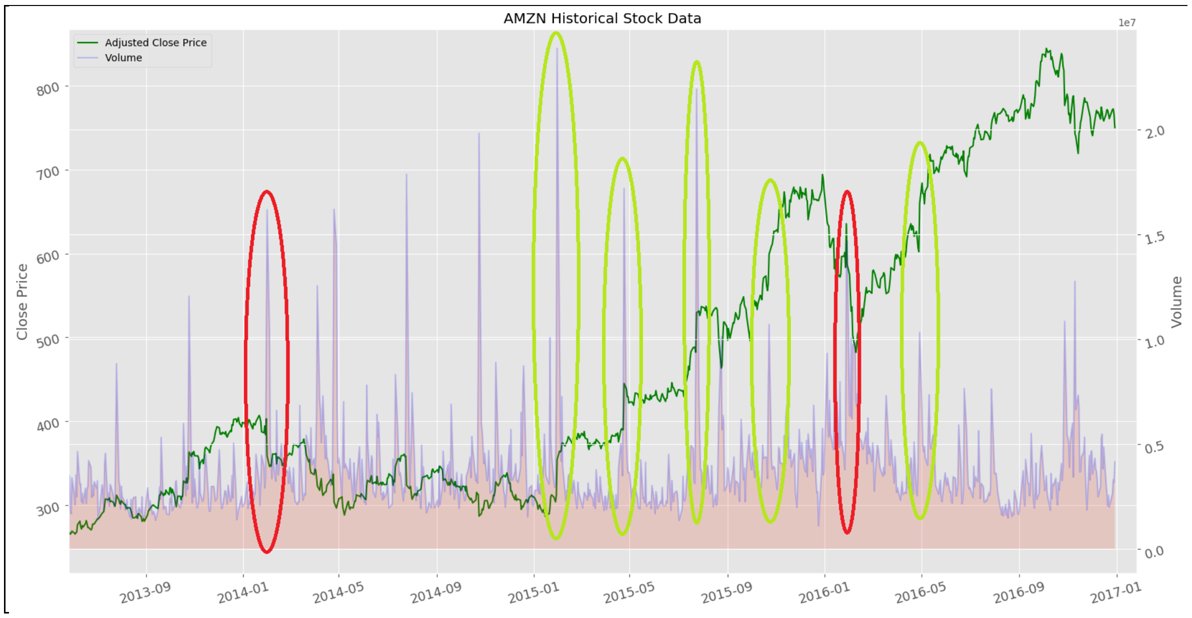


Figure 10 Historical stock prices for 'AMZN' with highlighted volume spikes

Reflection

In this project I achieved the goal of predicting the adjusted closing price for a variety of stocks four days into the future from a requested date. This was accomplished by creating a Graphical User Interface into which the user enters requested stock ticker symbols and dates. If the user does not know the stock symbol, a button is provided that takes the user to Yahoo! Finance where the user may look it up. This website also provides the historical stock price data that will be used by the Support Vector Machine model to make the predictions.

Currently, the user must download the data from Yahoo! Finance manually, but this is only due to a very recent change in the websites' structure. Once the necessary Python libraries update to account for this change, the GUI will automatically download the data and store it locally on the user's computer.

The solution imports the historical stock price data into a dataframe that it manipulates to accomplish its following tasks. It takes this dataframe and adds more columns to it using basic mathematical operations. These new columns are the features and labels to be used by the SVM. Then the application splits the dataframe into training and testing dataframes with sizes that differ based upon the dates that the user requests. Next, backfilling and forward-filling are applied to the dataframes to eliminate any ‘NaN’ values. The SVM then takes these dataframes and fits itself to them. Using a range of parameters, the model tunes itself and uses the optimal parameter to make the requested predictions. Finally, the predictions and their percent errors of both the SVM and the benchmark models are gathered together with other pertinent information and presented to the user in a results table.

An aspect of the project that I found interesting was making the Graphical User Interface and building all of the features into it. I have never built one before, and it was exciting to go through the process. Working with data, code, and Machine Learning libraries is fascinating to me, but having this extra visual layer built on top of it all really made it come to life. Adding the ability to graph stock price data directly from the GUI was also very stimulating for me, and I think will be for the user as well.

This project was challenging in multiple ways. It was formidable to go through all of the steps of data acquisition and manipulation. Researching the numerous Machine Learning algorithms and seeing which ones would be the best-suited for this kind of application took a lot of work in itself. Once I had a short list of algorithms to try, a lot of effort went into making predictions with them, using different combinations of parameters. Building the GUI was a new kind of problem for me to solve in itself- and it took quite a while to complete- but a very worthwhile one.

As discussed in the ‘Results’ section of this report, the final model produces price predictions for various stocks with less error than the benchmark for a date four trading days after the requested date. Because of this, the model meets the expectations set for this project.

However, this model should not be used to make price predictions for all number of days into the future. Also shown in the ‘Results’ section, the average error produced by the Machine Learning model was higher than that of a Simple Moving Average with windows of two, six, and eight days.

Improvement

One way that this model could be improved is to do more feature engineering or parameter tuning. For example, in the ‘Free-Form Visualization’ section above, a correlation was shown between spikes in the volume of shares traded and a significant increase or decrease in share price. This may help the model to make predictions with less error than a Simple Moving Average for periods of two, six, eight, or more days into the future.

The application itself could be improved to automatically download necessary historical stock price data to be used for predictions. The Python libraries used to do this kind of task should update at some point to accommodate the structure changes made to the Yahoo! Finance system.

The application could further be expanded to store portfolio information created by the user. If the user buys shares of a stock at a particular time, they may want to store that information. Based upon predictions made by the SVM model, the user may want to sell that stock before an expected loss is incurred, or after a gain is made.

It is common knowledge that people investing in stocks should not buy stocks in only one company, but multiple companies. This is to protect against the possibility of any one company dropping sharply in value, losing lots of money for the user. Risk/reward features could be built into the application to help the user diversify funds safely by investing more in safer companies, and less in riskier ones.

Works Cited

"1.11. Ensemble Methods." *Scikit-learn: Machine Learning in Python*, scikit-learn.org/stable/modules/ensemble.html#forest. Accessed 22 June 2017.

"1.4. Support Vector Machines." *Scikit-learn: Machine Learning in Python*, scikit-learn.org/stable/modules/svm.html#svm-regression. Accessed 14 June 2017.

"1.6. Nearest Neighbors." *Scikit-learn: Machine Learning in Python*, scikit-learn.org/stable/modules/neighbors.html#regression. Accessed 22 June 2017.

"3.2.4.3.2. Sklearn.ensemble.RandomForestRegressor." *Scikit-learn: Machine Learning in Python*, scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html#sklearn.ensemble.RandomForestRegressor. Accessed 13 June 2017.

"Adjusted Closing Price." *Investopedia*, www.investopedia.com/terms/a/adjusted_closing_price.asp. Accessed 8 June 2017.

Brownlee, Jason. "How to Tune Algorithm Parameters with Scikit-Learn." *Machine Learning Mastery*, machinelearningmastery.com/how-to-tune-algorithm-parameters-with-scikit-learn/. Accessed 14 June 2017.

Kim, Kyoung-jae. "Financial time series forecasting using support vector machines." *Neurocomputing*, vol. 55, no. 1-2, 13 Mar. 2003, pp. 307-319.

"Look-Ahead Bias." *Investopedia*, www.investopedia.com/terms/l/lookaheadbias.asp. Accessed 22 June 2017.

"Sklearn.neighbors.KDTree." *Scikit-learn: Machine Learning in Python*, scikit-learn.org/stable/modules/generated/sklearn.neighbors.KDTree.html. Accessed 19 June 2017.

"Sklearn.neighbors.KNeighborsRegressor." *Scikit-learn: Machine Learning in Python*, scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsRegressor.html#sklearn.neighbors.KNeighborsRegressor. Accessed 19 June 2017.

"Simple Moving Average - Technical Analysis." *OnlineTradingConcepts.com*, www.onlinetradingconcepts.com/TechnicalAnalysis/MASimple.html. Accessed 5 June 2017.

"Volume." *Investopedia*, www.investopedia.com/terms/v/volume.asp. Accessed 8 June 2017.