Capstone Project – Predicting Stock Prices

Collin Hazlett

7/30/2017

Project Overview

Ever since the Stock Market was created, people have been trying to make money by predicting the future price of a stock but it hasn't always turned out well. It has amazed and confounded the greatest of minds. Even Sir Isaac Newton when speaking of the stock market, said that he 'could calculate the motions of the heavenly bodies, but not the madness of the people'. Thousands more traders have given their entire careers to predict the stock market in the hopes of massive profits or at least a better understanding. With the advent of machine learning we can outsource our decisions try to make better predictions and strategies.

In this project, I have created a stock prediction algorithm that is capable of automatically selecting relevant features from a set of stock prices and optimizing its own parameters to provide an estimate of future stock price at any time into the future the user chooses. This idea was selected from the sample project ideas given by Udacity as part of this Nanodegree.

Problem Statement

The goal is to generate future stock prices through the following tasks

I. Build a data set of stock prices from Quandl of the 500 stocks which are currently part of the S&P 500
II. Implement a user interface which allows the user to select a stock, training range, and how many days into the future they would like to predict
III. Automatically select relevant feature for our model. Features in this project are other Stocks within the S&P500 and other calculated fields from the target stock price.
IV. Train multiple models with cross validation to select important parameters and optimize the models
V. Once Trained, an ensemble effect will be implemented to increase the validity the predictions from all models
VI. Predictions will be plotted and information about the model is presented to the user

The final output is expected to help the user decide whether to buy a stock.

Metrics

R-squared is a very common and widely used metric to judge a regression model. R-squared is calculated as follows:

$$r = \frac{n(\Sigma xy) - (\Sigma x)(\Sigma y)}{\sqrt{[n\Sigma x^2 - (\Sigma x)^2][n\Sigma y^2 - (\Sigma y)^2]}}$$

In Sklearn regression models, the default scoring method is already defined as R-squared so there will not be a defined scoring algorithm to calculate r-squared, but instead will be used with the .score(X,y) format.

In addition to R-squared, the mean absolute error will also be used as a metric so the user can easily understand how to interpret the results of the model in terms of US dollars. The mean absolute error is given by:

$$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| = \frac{1}{n}\sum_{i=1}^{n}|e_i|$$

Where

$$AE = |e_i| = |y_i - \hat{y}_i|$$
$$Actual = y_i$$
$$Predicted = \hat{y}_i$$

Because this program has the capability to predict any stock price any number of days into the future, the score of the model will greatly depend on the inputs given by the user and relative accuracy compared to the benchmark model.

Analysis

Data Extraction

Quandl is a website that maintains stock market data and provides a free API in which to pull data from. After pulling a list of stock tickers in the S&P500 from Wikipedia I borrowed code from pythonprogramming.net's financial data script to iterate through the list and pull [Date, Open, High, Low, Close, Volume, Ex-Dividend, Split Ratio, Adj. Open, Adj. High, Adj. Low, Adj. Close, and Adj. Volume] for each stock from 2000-01-01 to 2016-12-3131 which amounts to 4,032 trading days so the data set contains 4,032 rows.. I was able to pull 503 csv files and save them locally on my computer so that I wouldn't need to query Quandl each time I needed stock data. From these files, I then built master csv containing only the [Adj. Close] from each stock. This master file was then used as my main dataset which the project is based.

The master csv file is named “sp500_joined_closes” and contains 504 Columns and 4278 rows with row 1 as the column names, and column 1 as the [Date] field.

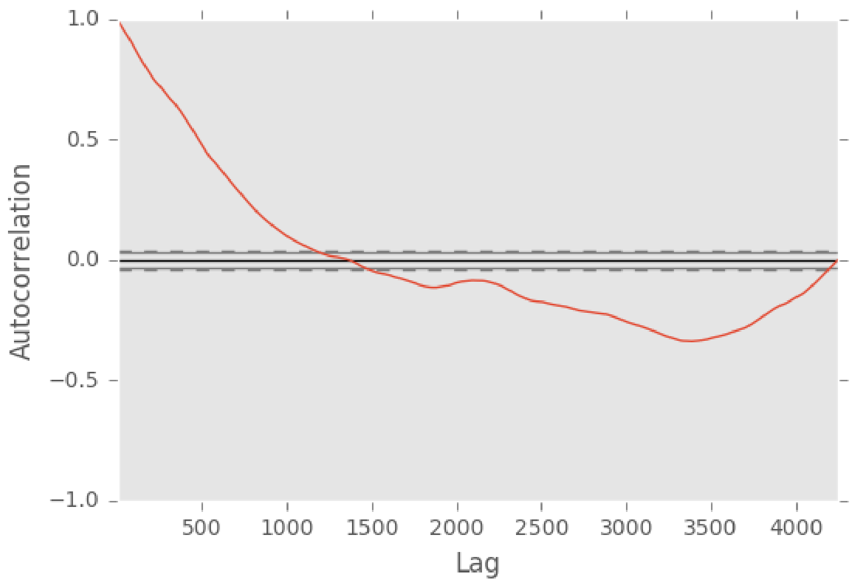
Because the S&P500 is a list that changes, not all stocks have data spanning the entire date range from 2000-01-01 to 2016-12-31. In order to create a complete DataFrame in pandas, I used “dropna()” in order to drop any columns that contained missing values. This brought the final data set to **387** full columns.

Data Exploration

The stock market is not open on the weekends so when exploring the [Date] field, you will notice that weekend dates are not included in the data. Because of this, I chose to not use [Date] as an index and instead create a [Day_N] field which would count the number of days of actual trading that had taken place since the 2000-01-01.

I decided to treat this project as a normal regression problem instead of a time series since I felt more comfortable with the processes we learned during the course.

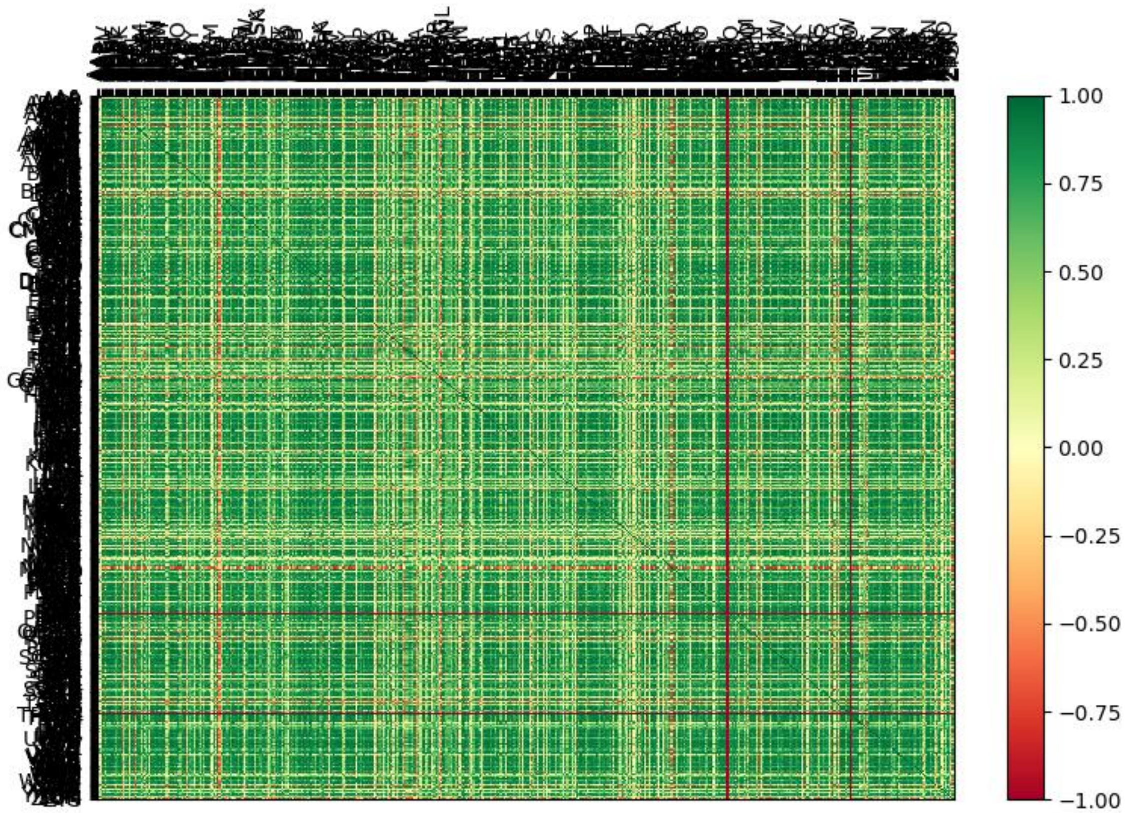
According to the efficient market hypothesis, the current price of a stock should account for all knowledge about the stock’s history and expected future and should value to the stock accordingly. If this was the case, we would expect that tomorrow’s price should be very similar if not the same as today’s price. Below is an autocorrelation graph for the stock price of 3M (ticker = “MMM”). As you can see, the current stock price is highly correlated with its own historic prices but in a decreasing manner, i.e. the more time into the past we look, the less correlated the prices become. Therefore, as we look farther into the future we will need other variables in our model to supplement its accuracy.

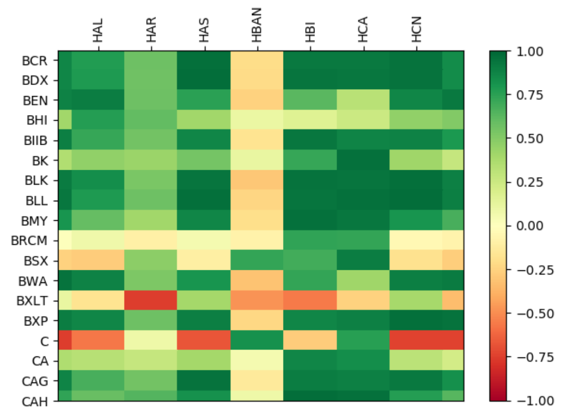


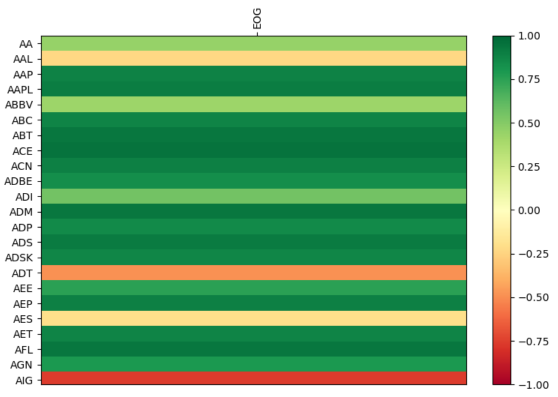
When first investigating the dataset, you will notice that the sheer size incapacitates a human's ability to understand the relationships in the data. Normally a good place to start would be with a correlation matrix like the image below, but due to the size of the data set, this would require a huge amount of time and energy from the user in order to find which variables need to be added to our model.

The first image below, is a correlation matrix which has been color coded to show on a scale from green to red how correlated each stock is to all other stocks. With such scale, the axis labels are incomprehensible. The next two images, show the matrix after zooming in to investigate. Even with this zooming ability the user loses a full picture and the ability to grasp any pattern.

For this reason, I will need to rely on the computer to automate the selection process for me.







Benchmark

The benchmark for this project will be the accuracy from a Univariate Linear Regression with current stock price as the X and future stock price as the Y with no feature scaling. This will ensure an appropriate benchmark for each Company the user picks and for any number of days into the future the user decides.

This model will run each time the script is run so that we can judge our new models against the correct benchmark according to the user inputs. Both R-squared and Mean Absolute Error are calculated for the user to compare against.

Algorithms and Techniques

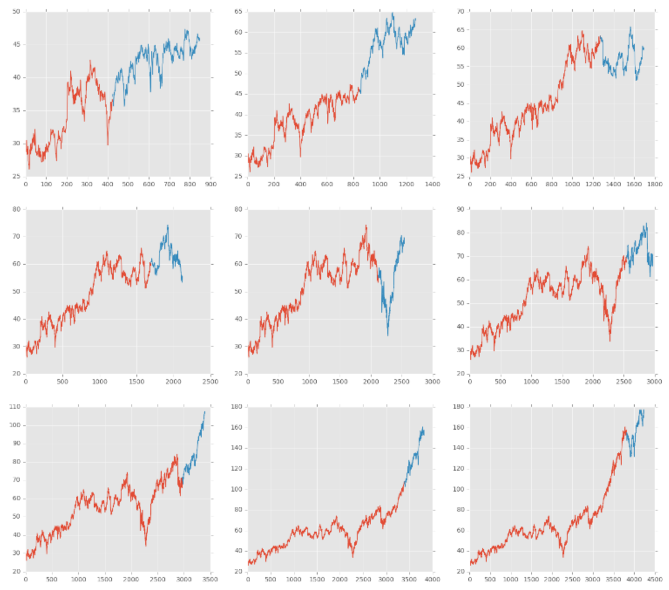
Feature Scaling

Because these stocks have such wide-ranging prices, feature scaling was important to carry out before the feature selection and model development stages. The StandardScaler() function standardizes the features by removing the mean and scaling to unit variance. According to sklearn documentation, centering and scaling happen independently on each feature by computing the relevant statistics on the samples in the training set. Mean and standard deviation are then stored to be used on later data using the transform method.

Cross Validation

For parameter optimization, I used the cross-validation algorithm “TimeSeriesSplit” which splits the training set into 10 different portions sequentially expanding the window of time to be trained over but keeping the testing portion the same size. Because the data is time series in nature we cannot randomize the samples, so the testing portion must come directly after the training portion.

Below is an example of the segments created by the “TimeSeriesSplit” technique. The red data is used for training while the blue is for testing.



Feature Selection

Feature selection will play a major part in the overall algorithm. Due to the number of variables in the data set, it will be very important to narrow down on the most important variables. There were 2 techniques which are used in my process. In this way, I created 2 smaller and different datasets on which to train and pass multiple models through.

The first is Principle Component Analysis; which will take in all variables in the full dataset and finds the number of principle components based upon the amount of explained variance added at each new principle added. When the added explained variance drops below 0.01, the selection process will end and the number of components chosen will be the previous number of components. This allows the user to remove himself from the process and automates the decision.

The second technique used is Recursive Feature Elimination with a Linear Regression model used as its base model. This technique starts out using all variables in the chosen model and then iteratively removes features that perform the worst in the model. The specific algorithm chosen used cross validation as defined above to select variables. The function I have created will return a list of the top 20 features from our data set.

Regression Algorithms

As stated above, because we are predicting a quantity, the problem requires a regression estimator. The models I have selected are: Stochastic Gradient Decent Regression and LASSO Regression, Support Vector Regression, and K-Neighbors Regression. These were chosen from the “scikit-learn algorithm cheat-sheet” after the flow of decisions was followed.

LASSO Regression

I am very familiar with simple Linear Regressions but wanted to learn more about the LASSO Regression by using it. The LASSO model is good when the it is important to only have a few variables because it rewards the model with the L1 regularizer. According to sklearn, the objective function to minimize is:

$$\min_{w} \frac{1}{2n_{samples}} ||Xw - y||_2^2 + \alpha ||w||_1$$

The lasso estimate thus solves the minization of the lease-squares penalty with $\alpha ||w||_1$ added, where α is a constant and $||w||_1$ is the ℓ_1-norm of the parameter vector.

Stochastic Gradient Decent Regression

Even though the SGD Regressor is stated to be best when used with more than 10K samples (according to sklearn), I wanted to learn about its process and to compare it to other models in our algorithm. The SGD implements a plain stochastic gradient decent learning routine which can support different loss functions and penalties to fit linear regression models. The two loss functions passed in as parameters were "squared_loss" for ordinary least squares and "huber" for robust regression.

Support Vector Regression

Support Vector Regressions was selected because it has the capacity to use different kernels to construct hyper-planes to transform the data. The two parameters to tune are C, the penalty term, and epsilon, which determines when a penalty is given. Although present in the code, this model has been deprecated from the final script because it took too long to train and performed very poorly on all stocks tested on.

K Neighbors Regression

KNeighborsRegressor was chosen because it's non-parametric and can work on non-linear data. However, forecasting stock prices into the future is difficult and performs badly if the price is increasing into a new range. After realizing this miscalculation, the model is only used as an ensemble method.

Neural Network (MLP Regressor)

A neural network was chosen as an additional ensemble method after reading this article https://www.analyticsvidhya.com/blog/2015/08/optimal-weights-ensemble-learner-neural-network/ . This network will be able to tune the weights applied to the different model predictions to find the best combination and should be able to increase the accuracy over the naïve average ensemble method.

Methodology

Building the Dataset was done in two steps. First a list of the stocks included in the S&P500 was pulled from Wikipedia. Then data was pulled from Quandl's API by iterating through the list. Each stock was pulled in separately and saved in its own csv file. Once all data was pulled, it was amalgamated together by pulling only the [Adj. Close] from each file and joining on the common [Date] field. This process created a "sp500_joined_closes.csv" which is used to store the data and is read in as a pandas data frame each time the scrip is run.

Dataset cleaning and processing was needed before any analysis was run. Although the data was clean on an individual basis, not all stocks were live during the dates selected so when looked at over the full range of data, we could see NULL values before the company went public. Dropna() was used to drop any column which did not have a full set of data over the given date range. This produced a solid data frame with no NULL values.

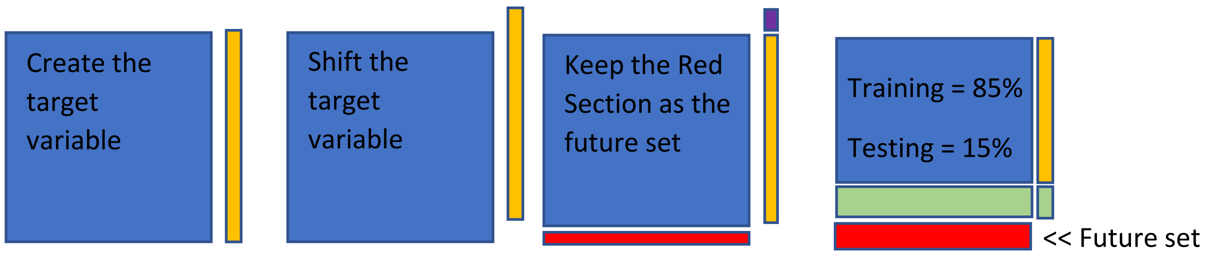
As stated above, this problem was not treated as a timeseries method but I still wanted to have "time" incorporated into the model. To accomplish this, [Day_N] was created by taking the index of the data frame as its value. This created a constantly increasing numerical variable with step size 1 and could help account for any left-over time trends if selected in the feature selection process.

The actual [Date] field is kept as its own variable throughout the script to keep track of dates of predictions and for chart labeling.

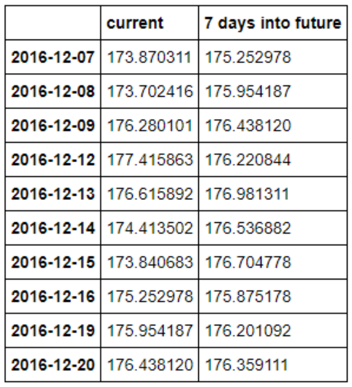
As part of the defined criteria for this project, the user had to choose certain parameters for the model. These were the date range for training, the number of days into the future to predict, and of course the ticker which is the target for the model. Currently, the script uses an 85/15% train test split but lines of code to enable the split by date if the user un-comments them out.

Calculated variables were chosen after completing the Machine Learning for Finance mini course on Udacity. Rolling Averages for 10, 20, and 30 days were added to the dataset but with the added fields, we lost 30 days' worth of data since each rolling average requires the number of days before it before it can calculate its value.

The target variable is created by selecting the stock price based upon the user's input and shifting the entire field up the number of days that the user wants to predict. For example, if the user selects 'MMM', a copy of this stock price will be designated as the target variable, then the entire target variable will be shifted up the number of days also designated by the user, say 7. This final set will be broken into training and testing sets based on an 85 – 15% split rate. Due to the shifting of the target variable, a third set will contain N number of rows depending on what the user defines as his future price to predict. These last rows will not allow for verification of our predictions so they will be kept by themselves as a "future" prediction set. Below is a visual representation of the process.


Create the target variable
Shift the target variable
Keep the Red Section as the future set
Training = 85%
Testing = 15%
<< Future set

The image to the right shows the current price and target future price as the current price after being shifted 7 days into the future



	current	7 days into future
2016-12-07	173.870311	175.252978
2016-12-08	173.702416	175.954187
2016-12-09	176.280101	176.438120
2016-12-12	177.415863	176.220844
2016-12-13	176.615892	176.981311
2016-12-14	174.413502	176.536882
2016-12-15	173.840683	176.704778
2016-12-16	175.252978	175.875178
2016-12-19	175.954187	176.201092
2016-12-20	176.438120	176.359111

TimeSeriesSplit was selected for cross validation across the training set, and broke the time frames into 10 different sets with an expanding window as described above.

Due to the substantial number of variables in the dataset, feature selection was an important part of the problem. As stated above, both PCA and Recursive Feature Elimination were both used to create 2 different training sets on which to train our models. These 2 sets then allowed each model to be trained over multiple variables and add to our ensemble method at the end.

1. The PCA selection process was designed to select the optimal number of components and then transform the data into that number of variables.
2. The RFE process will select the top performing features when scored in a linear regression and then return the list of variables.

The general learning process proceeds like so:

1. A function for training a specific model is defined
2. A dataset is passed to the training function
3. TimeSeriesSplit() splits the data into 10 different subsets on which to perform cross-validation
4. The learning algorithm is defined
5. A dictionary of relevant parameters and their values is defined
6. The regressor, parameters, and cross-validation set are passed to a GridSearchCV() function
7. The training and target sets are passed into the grid
8. The best estimator is returned

During the model building process, multiple other regressors were tried. Support Vector Regression, K-Neighbors Regression were both tried but eventually dropped because their scores were very low during the trial process. A Random Forest Regressor was also planned into the project but after inferior performance, it was also dropped.

Refinement of the model was an adaptive process that started out by exploring the different models and their parameters. I primarily used a Jupyter Notebook to visualize the process and created charts for predictions and scores for feature selection. After understanding the process for feature selection in this way, I thought it best to automated the algorithm by having it choose the best number of features. Given that the target stock could potentially change each time the script is run, there was a necessity of taking the user out of the optimization of the algorithm as this would add extra time unnecessarily. Grid search was then chosen to refine the models automatically based upon the parameters passed to it.

After all models are scored, the ones which performed the worst are excluded going forward. The decision process only selects models with positive R-squared. This is done so that the ensemble models are not held down by the worst models.

Multiple ensemble regression techniques are then implemented. The average of the model predictions, a neural network, a linear regression and a KNR model were all employed separately. This process produced a much better result than any of the other models on their own.

Users Guide

Currently the user experience for this script is elementary. It requires the user to open the script, and change the top variables before running. The script was designed using SublimeText3 which does not allow user input in the command line which is why a rawinput() function is not used. In future versions of this script, an interactive Jupyter Notebook or another program like Tkinter will allow for a better user interface.

Below is an example of the format that the variables mush be in:

ticker = 'MMM'	a string containing the ticker from the S&P500 which has price data from 2000 to 2016
days = 7	an integer greater than 0
start_date = '2000-01-01'	a date string in the form YYYY-MM-DD
end_date = '2016-12-30'	a date string in the form YYYY-MM-DD

**Currently, the start_date and end_dates are not being applied to the script as input variables. By un-commenting a few lines of code, this can easily be changed, but I felt that it made the user experience more confusing in the current format.

After defining the variables above, the script can be run and an output will show the selected features and scores for the models run. A graphical printout will also be presented which shows the stock price over the training range, the stock price and predicted price over the testing range, and a predicted future stock price over the range where no verification data is present. (See below image in the Conclusion)

Model Evaluation and Validation

During the process of model implementation, it was found that certain algorithms performed very badly and were discontinued. These models continuously produced prediction scores over the testing set that were negative. For an R squared value to be negative, it must essentially be perpendicular to the data and a completely horizontal line would predict the target price better. The models for which this occurred were: K-Nearest Regression, Support Vector Regression with kernels for ‘rbf’, ‘linear’, and ‘poly’, Decision Tree Regressor, and Random Forest Regressor. Once adequately performing models were found, they were selected to be used in the final script. The original plan was to create multiple regression models and average the outputs together but since only 2 made it through the selection process, multiple training sets were presented to achieve the desired effect.

Because the testing set is so large, I do believe that the model can be trusted at its given score level. The script can be used to predict many different stocks and different points in the future, so it is hard to confidently say whether the model is accurate for anything the user wants. However, the decision processes and models in place are clearly documented and are generally understood by most machine learning practitioners. This should allow for an easy explanation to the user at the end of the script.

A linear regression with only current price as a variable and no feature scaling was chosen as the benchmark model for our analysis as this seemed the simplest model. In the tests run, the benchmark model performed better than any of the other models developed in our process. This leads me to believe that a different approach would be needed in the future.

Justification

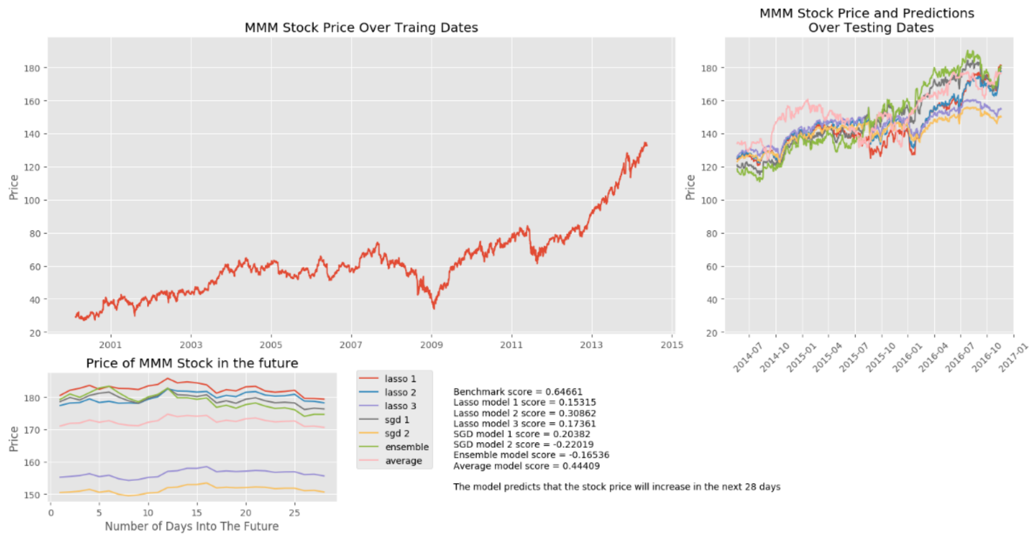
Although much depends on the chosen stock and number of days in which to predict, my testing of the script showed that the score for any one of the models was either not much better than the benchmark or performed worse than the benchmark. Even when an ensemble method was implemented to see if we could improve my model’s score that way, the ensemble model was only held down by the lesser performing models and produced a score that was lower than the top performing one. Overall the Neural Network ensemble produced scores higher than the other ensemble models.

I believe that my model is performing correctly but due to the competitive nature of the stock market, the model did not find a better method of predicting with such a simple algorithm. The market dynamics ensure that any price expectations are already considered when someone buys, therefore since the current price is already such a good predictor, I would guess that other people have justified using similar techniques already.

Conclusion

Free-Form Visualization

Below is an example print-out from the script when run on the MMM ticker with a 28-day prediction. You can find this image in my GitHub in the "images" folder.



```
Benchmark score = 0.64661
Lasso model 1 score = 0.15315
Lasso model 2 score = 0.30862
Lasso model 3 score = 0.17361
SGD model 1 score = 0.20382
SGD model 2 score = -0.22019
Ensemble model score = -0.16536
Average model score = 0.44409

The model predicts that the stock price will increase in the next 28 days
```

This script for predicting stock prices produces outputs that are somewhat in line with the actual price of the stock so it does give the user useful information about the future although it would need more in-depth analysis before anyone tries to use it to make money in the stock market.

Because I created a dynamic pipeline that filters out bad models and keeps only the good ones, the models chosen can change from one stock to the next. The trends within the data greatly affect the outcome of models so those stocks which have linear trends will have better performing linear models but in more complex data, the PCA models perform better.

During my conversation in my 1 on 1 meeting with a Udacity Mentor, we discussed the idea that this script could be utilized as a stock selection mechanism by iterating through each stock and then finding the ones where the model performed the best. This however would require much more processing power which is better suited for Cloud computing. This might be feasible in the future but at the moment, this would take too much time on my own person laptop.

Reflection

During my project, I accomplished the following:

1. **A dataset was created out of S&P500 stock prices from 2000-2016**
2. **Data was processed into a usable format with calculated fields add**
3. **The user can choose the desired stock ticker and number of days into the future to predict**
4. **Feature scaling and selection narrowed down the number of variables from 300+ to 6**
5. **Models were designed, trained over 85% of data with parameters optimized through cross-validation**
6. **Predictions were made with multiple models and data sets**
7. **Ensemble modeling was performed**
8. **A graphical representation of the training, testing, and future sets were presented along with model and benchmark scores for user guidance.**

Although my model does not perform any better than a simple linear regression on the stock price against its future price, the methods in place did seem to work properly and allowed myself to learn how much planning goes into a full project. I got to implement new types of algorithms and practiced drawing out my project beforehand.

I was very proud of myself with the output I designed and how much I learned during the entire process. It took me multiple iterations to fix bugs in the code, format the code, format the printout.

Improvement

Although the user can make choices for the model, it is not designed very effectively. I have imagined two ways to improve the user experience; first would be to utilize a Jupyter Notebook with dynamic cells which can allow the user to choose options from a drop-down list or scroll bar before running the script. The second, more difficult way, would be to design a dashboard with Tkinter with buttons and options for the user to choose. This would require much more effort as I have not used Tkinter before. Of these two I believe that the Jupyter Notebook would prove the most effective.

Other improvements could be made by running the stock predictor over all the stocks to find the stocks which the model fits the best. This could turn the predictor into a stock selector. A more capable computer or Cloud service would need to be utilized for this to be feasible.

In future iterations of this project, I would like to try different timeseries or deep learning algorithms to see how they would perform over the same data.

In projects after this one, I would like to better organize and streamline my code by using pipelines and iteratively cycling through different models.

Source Code

The code for this project can be found on my GitHub here:

Project Folder https://github.com/Mathtodon/Udacity_Project

Script to be run with Python 2: Stock_Price_Predictor_py2.py

Script to be run with Python 3: Stock_Price_Predictor_py3.py

File containing defined models: models.py

Images of output can be found here:
https://github.com/Mathtodon/Udacity_Project/tree/master/images

I would like to thank the following sources for their code which was repurposed during my project.

Pythonprogramming.net - Python Programming for Finance videos and code

Harrison's videos and code helped me create the dataset and visualize the correlation matrix

https://pythonprogramming.net/getting-stock-prices-python-programming-for-finance/

Diving into data – a blog on machine learning, data mining and visualizations

The pretty_print_linear() function was very helpful to understand the models

The feature selection process helped me build my own models

http://blog.datadive.net/selecting-good-features-part-ii-linear-models-and-regularization/

UDACITY –

Learning models from the housing price prediction project were repurposed as a guide for models defined in the models.py file