Capstone Project Tommy Li

Machine Learning Engineer Nanodegree September 30, 2016

Build a Stock Price Indicator

Definition

Project Overview

A python script that takes a list of ticker symbols and predicts their adjusted close prices, closing price adjusted for stock splits and dividends, based on their real-time open price, highest price, and trading volume. This script trained with the past data from Quandl and predicts the daily adjusted close price with the real-time data from Yahoo Finance.

Since the adjusted close price varies based on how people trade, choosing which algorithm to use is the most challenging part of this project. There's no perfect algorithm for this problem because no algorithm can read people's minds. However, like least recently used is a robust cache algorithm, people tend to trade similarly to their recent past. Therefore, the best we could do is to predict based on the recent past data.

Problem Statement

The goal for this problem is to predict the adjusted close price for all the given stock. These steps include:

1. Read all the inputs and setting from input.txt
2. For each ticker symbol, query its past and real-time data based on the setting.
3. Preprocess the data and create the train and test sets using cross-validation.
4. Train the data using bagging regressor with tuned parameters.
5. For each ticker symbol, predict and print out the adjusted close price using user's data or real-time data from Yahoo Finance with the trained regressor.

Lastly, if the user wants any recommendations, plots, or additional predictions, the indicator will print out that information along with the adjusted close price.

Metrics

Root mean squared error is a common way to calculate error metric. It is useful to measure error for this problem because it punishes large errors. Root mean squared error defined as:

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

Here, n is the number of test data. Y is the true value, and y hat is the predicted value. Thus, I define root mean squared error as my error range to give a sense of variation for the predictions.

Analysis

Data Exploration

The train and test data for this project are queried from Quandl. Quandl has most of the stock data from the past few years. Since the stock market has 252 trading day per year on average, we will have at least 700 data for most stock. Thus, there will be enough data for us to train and test for most stock unless the stock recently joined the market.

The data from Quandl has datetime index with 6 features:

- 'Open': The opening price for the day
- 'High': The highest traded price for the day
- 'Low': The lowest traded price for the day
- 'Close': The closing price for the day
- 'Volume': The number of stocks was traded for the day
- 'Adjusted Close': The closing price adjusted for stock splits and dividends for the day.

Since we are predicting the adjusted close price, 'Close' feature should not be included because we can't obtain this information until the end of the day.

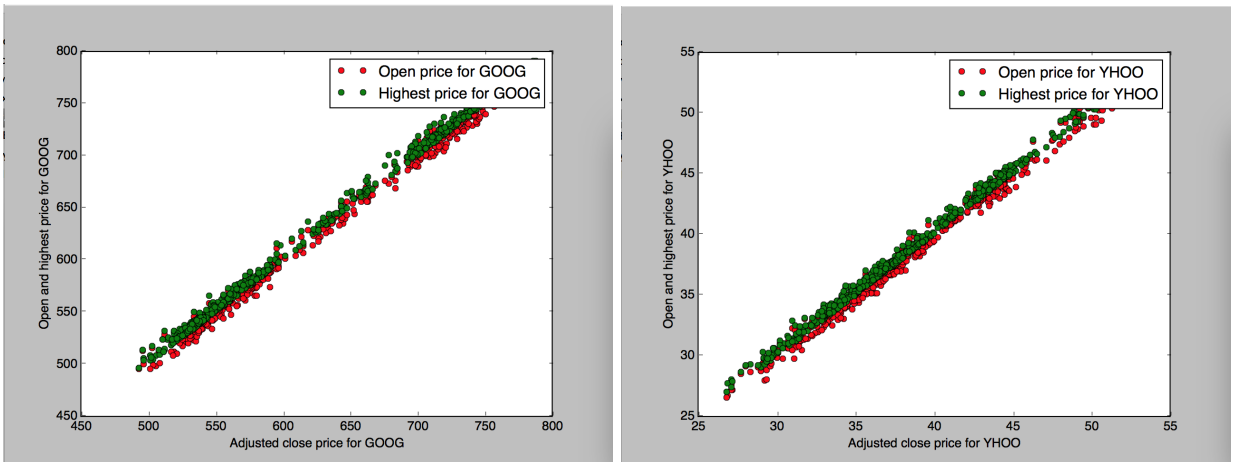
Although ideally, we want to train data based on each time interval of the day to give a better estimation, the free data we have only contain the highest and lowest price for the

whole day. Therefore, the adjusted close price is always between the highest price and lowest price of the day in our data. Since this project is for predicting the adjusted close price during the middle of the day, we only can obtain information about the temporary high, low, and volume of the day. Hence, including both high and low feature will limit the predictions between the temporary highest and lowest price which is not always true.

However, we still need either the ‘high’ or ‘low’ feature to give the learner some information about whether the stock tends to grow or drop. For example, if the temporary highest price of the day is close to the open price, its close price will tend to be lower than or stay the same as the open price. Thus, by removing the ‘high’ or ‘low’ feature, it gives a larger range for estimating the price which has higher possibility to predict accurately. As a tradeoff, the estimator will either overestimate or underestimate the actual close price. Overestimating will encourage users to invest and lose money where underestimating will prevent users to invest and miss an opportunity. Because of that, removing ‘low’ feature will predict a better adjusted close price since it will have a tendency to underestimate which has the lower risk for users.

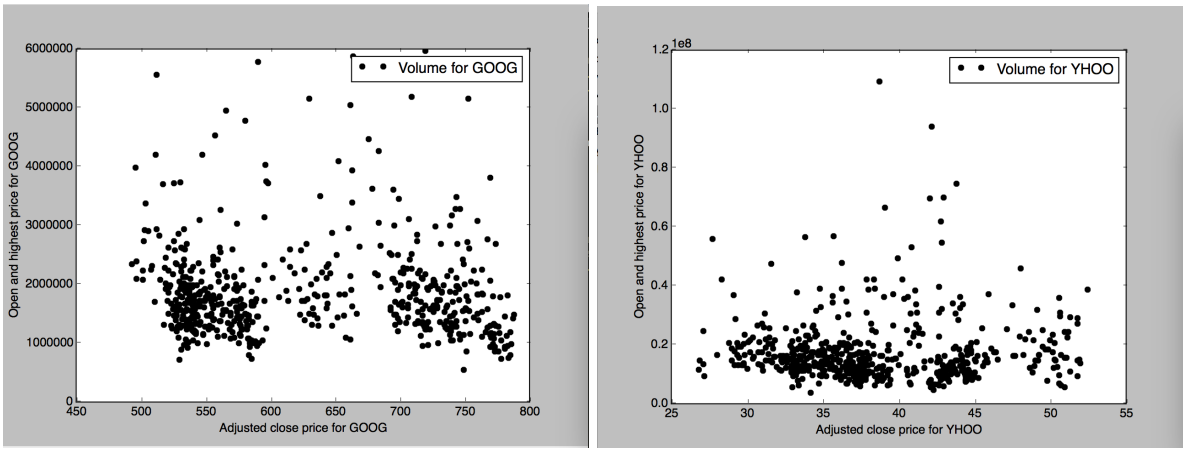
Exploratory Visualization

The two plots below show the correlation between the adjusted close price with ‘Open’ and ‘High’ features for Google and Yahoo.



We can clearly see that there’s a strong correlation among the ‘Open,’ ‘High,’ and ‘Adjusted Close’ features. Thus, ‘Open’ and ‘High’ features are very useful for predicting adjusted close price.

In addition, the two plots below show the correlation between the adjusted close price and ‘Volume’ feature for Google and Yahoo.



Although there is no strong correlation for these two plots, we can see that most data have similar trading volume. Therefore, if the real-time volume is close to the average daily volume, the adjusted close price won't vary as much from the real-time price because there are not much trading left in general. Hence, 'Volume' feature gives information about how much the adjusted close price could vary from the real-time price.

Algorithms and Techniques

Since this is a regression problem, decision trees and k-nearest neighbors regressor will be the right choices because stock prices are changing by trading. Moreover, Humans tend to do similar trading from the past unless there is some current news that influences the stock market. Thus, decision trees and k-nearest neighbors regressor will well capture the past data and make a good prediction.

Decision trees will split the data into small groups based on maximizing information gain. Thus, these groups well capture the past data and precisely predict data that are similar to the past data. However, new data points or data that act entirely new from the past will not be captured in decision trees.

K-nearest neighbors will predict the result using k-nearest data points from the past data. Hence, it also can well capture the past data and assume similar data points will have similar results. However, similar to decision trees, it performs poorly on new, non-similar data points. Nevertheless, as I mentioned in the project overview section, people tend to trade similarly to their recent past. Thus, decision trees and k-nearest neighbors should perform well for this problem.

Although the parametric algorithm is better for capturing new data points, the nonparametric algorithm captures the old data points more precisely. Since the primary goal of this project is to estimate the unseen data from next day, data points for the next day will be similar to the old data points unless a rare, significant outlier exists. Since trading stocks are risky and those outliers will be part of the risk, I decided to use the nonparametric algorithm to give a better prediction on average.

Furthermore, ensemble methods with these two base algorithm can enhance the prediction even better. Therefore, I use the bagging regressor for my ensemble method because it creates more samples by training multiple subsets of data. Since some stocks have limited data, and recent data are always better than old data, bagging will do a great job to make the best decision will a decent data size.

The bagging regressor will train multiple dataset sizes to encounter overfits and bias. Then, create a train and test set using cross-validation and fit it to the bagging regressor. We will use the grid search to tune our models since we don't have a clear idea what parameters are the best. We will tune the maximum depth for the decision trees, the number of k neighbors for k-nearest neighbors, and the number of estimators for bagging regressor with a random state. Lastly, we will decide the best dataset size and parameters based on the lowest root mean squared error we get.

Benchmark

We want to make sure the root mean squared error is within 2 percent of the current price. Furthermore, since the best way to verify our prediction is to compare with the actual result, next day's prediction should within 2 to 3 percent variation of the actual adjusted close price. Although this project only forecasts next day's adjusted close price, I will also test unseen data from a week later to verify its consistency. Lastly, the indicator should come up accurate predictions for most common stocks, so I will use some popular stocks such as Google and Yahoo to justify our prediction.

Methodology

Data Preprocessing

The preprocessing is done by 'data_preprocessing()' function in learner.py. The preprocessing contains the following steps:

1. Take the data and calculate the differences between the open and adjusted close price for each day.
2. Calculate the interquartile range(IQR) for the differences and find all the outliers. Any data that falls below Q1 - 1.5*(IQR) and above Q3 + 1.5*(IQR) will consider as outliers.
3. Remove all the outliers and return the remaining data.

Since dramatic changes between the open and adjusted close price only happen with some big news, those outliers will strongly influence the result in k-nearest neighbors. Thus, remove those outliers would improve the accuracy of the prediction in general.

Implementation

The implementation is done with these three files: main.py, data.py, and learner.py. Also, users need to put all their inputs to inputs.txt.

Data.py
This file contains two functions which are used for query the past and real-time data from Quandl and Yahoo Finance. Thus, all the private API keys are stored in this file.

Learner.py
The file contains two big classes, trainer and predictor, for training and predicting the adjusted close price. The trainer class contains the following variables.

```
class trainer:
       def __init__(self, ticker):
              self.data = None
              self.ticker = ticker
              self.clf = None #Trained regressor
              self.clf_score = None #Regressor R2 score
```

The goals of this class are getting the data and finding the best regressor with its score. Declare trainer with a ticker symbol and call the ‘training()’ function in this class to accomplish all these goals. Training() will take a ticker symbol with a starting and ending date for the dataset along with some testing variables. Training() contains the following steps:

1. Query the data based on the given ticker symbol and dates. If there are no given dates, query the most recent n data (Will define n during refinement).
2. Preprocess and store the data to self.data.
3. Use cross-validation to create the train and test data.
4. Create the bagging regressor and tune its parameters using grid search.

5. With the best estimator, train it with the train data and calculate the root mean squared error with the test data.
6. Store the trained regressor and root mean squared error to self.clf and self.clf_score.

Then, declare predictor class with the ticker symbol and trained regressor and call pred_curr() to predict the adjusted close price. pred_curr() uses the trained regressor to predict the result based on some given 'Open,' 'High,' and 'Volume' features.

There is another function in predictor class called predicting(). This function will take a list of given dates from the past and make predictions based on the past data. This function is mainly for examining the accuracy for the trained regressor.

Main.py
This file acts the core part of this project. It calls the function run() and do the following steps:

1. Read inputs.txt and parse all the user inputs.
2. Create trainer and predictor for each ticker symbol.
3. Train each stock by calling trainer.training().
4. Query today's data and print out current price, temporary volume, high, and low.
5. Use the predictor to predict today's adjusted close price.
6. Print out the adjusted close price for each ticker symbol.
7. If user asks for recommendations, it recommends trading stocks that are above and below 1.5 times the error range.

Refinement

We can improve the current solution by choosing the best size of datasets and base estimator. Initially, I use bagging regressor with k-nearest neighbors as the base estimator to trained and tested with 200 data and get the following result.

```
$ python main.py
Developer mode on
Training for GOOG...
Average error range for GOOG: $25.3765
This is the result for GOOG:
Training data are from 2005-03-11 to 2016-09-22
Predicted adjusted close value for 2016-09-23 is 736.2161
Actual adjusted close value for 2016-09-23 is 786.9000
```

```
Training for YHOO...
Average error range for YHOO: $3.1031
This is the result for YHOO:
Training data are from 2005-03-11 to 2016-09-22
Predicted adjusted close value for 2016-09-23 is 33.9199
Actual adjusted close value for 2016-09-23 is 42.8000
```

We can clearly see that it has a high error range for both stocks with tuned parameters and the predictions are completely off from the true prices. Thus, k-nearest neighbors is not an excellent choice for the base estimator. In order to prove decision tree is a better base estimator, I compared both estimators with 200, 400, 600, 800, and 1000 data.

Error Range for	200 data	400 data	600 data	800 data	1000 data
GOOG(KNN)	$25.38 (3.23%)	$85.45 (10.86%)	$90.09 (11.45%)	$90.91 (11.55%)	$97.60 (12.40%)
YHOO(KNN)	$3.10 (7.24%)	$5.05 (11.80%)	$5.70 (13.32%)	$5.89 (13.76%)	$8.08 (18.88%)
FB(KNN)	$6.60 (5.16%)	$15.68 (12.25%)	$16.57 (12.95%)	$18.74 (14.65%)	$26.52 (20.73%)
MSFT(KNN)	$2.27 (3.95%)	$5.36 (9.33%)	$5.94 (10.34%)	$7.64 (13.30%)	$8.79 (15.31%)
GOOG(DT)	$4.83 (0.61%)	$5.02 (0.64%)	$4.54 (0.58%)	$5.52 (0.70%)	$51.38 (6.52%)
YHOO(DT)	$0.40 (0.94%)	$0.44 (1.03%)	$0.38 (0.89%)	$0.36 (0.84%)	$0.33 (0.77%)
FB(DT)	$0.81 (0.63%)	$1.00 (0.78%)	$0.78 (0.61%)	$0.94 (0.74%)	$0.84 (0.66%)
MSFT(DT)	$0.45 (0.78%)	$0.48 (0.84%)	$0.56 (0.98%)	$0.55 (0.96%)	$0.49 (0.85%)

(**The above results are based on all the data up to 2016-09-23**)

The above table shows that decision tree is much better than k-nearest neighbors for all dataset sizes. Thus, I decide to use bagging regressor with decision tree base estimator as our algorithm. Moreover, we can see that most stocks have lower error range on average with 600 data. Although ideally, the trainer can decide the best dataset size for

each stock by training subsets of data multiple times, the time complexity makes it cost too much, and it would take hours to find the best dataset size for all stocks. Hence, I set 600 as the default size and allow users to adjust it if they want. Now, here is the new result with 600 data that trained and tested with decision tree base estimator.

```
$ python main.py
Developer mode on
Training for GOOG...
Average error range for GOOG: $4.5411
This is the result for GOOG:
Training data are from 2005-03-11 to 2016-09-22
Predicted adjusted close value for 2016-09-23 is 782.9376
Actual adjusted close value for 2016-09-23 is 786.9000

Training for YHOO...
Average error range for YHOO: $0.3847
This is the result for YHOO:
Training data are from 2005-03-11 to 2016-09-22
Predicted adjusted close value for 2016-09-23 is 43.4275
Actual adjusted close value for 2016-09-23 is 42.8000
```

From the above figure, we can see that our new results improved by a lot. Moreover, the predictions and error ranges are within 2 percent of the actual results, so we also met the benchmark.

Results

Model Evaluation and Validation

At the end of the refinement section, we decided the best parameters with reasonable solutions. Here is the summary of the final default parameters:

- Algorithm: Bagging regressor with decision tree base estimator
- Data: Most recent 600 data from the Quandl query
- Regressor parameters: Tuned max depth and n-estimator with grid search. The possible max depths are 5, 10, 20, and 50. The possible n-estimators are 5, 10, 20, 50
- Preprocessing: Calculate the daily differences and eliminate all outliers from those differences.

These parameters are appropriate as we can see from the result in the refinement section. Moreover, according to the following figure, the model also generalizes well to unseen data. The training data are taken from 2005-03-11 to 2016-09-19 to ensure there is no data leakage. All the data from 2016-09-20 and beyond are unseen to the learner.

```
$ python main.py
Developer mode on
Training for GOOG...
Average error range for GOOG: $4.6457
This is the result for GOOG:
Training data are from 2005-03-11 to 2016-09-19
Predicted adjusted close value for 2016-09-20 is 768.9001
Actual adjusted close value for 2016-09-20 is 771.4100
Predicted adjusted close value for 2016-09-26 is 781.3202
Actual adjusted close value for 2016-09-26 is 774.2100

Training for YHOO...
Average error range for YHOO: $0.3575
This is the result for YHOO:
Training data are from 2005-03-11 to 2016-09-19
Predicted adjusted close value for 2016-09-20 is 43.1379
Actual adjusted close value for 2016-09-20 is 42.7900
Predicted adjusted close value for 2016-09-26 is 42.4244
Actual adjusted close value for 2016-09-26 is 42.2900
```

From the above figure, we can see that the predictions are still accurate with unseen data from the next day and next week. In addition, this model is robust enough because it produces consistent results for other stocks with training data from different time intervals(Will provide more details for this test in the justification section). Therefore, we could trust the results are within two to three percent variation.

Justification

We want to prove that this model is robust and satisfy with the benchmark. Thus, I tested the model with 2015's data from Facebook and Microsoft. (You can obtain the following results by replacing input.txt with test.txt.)

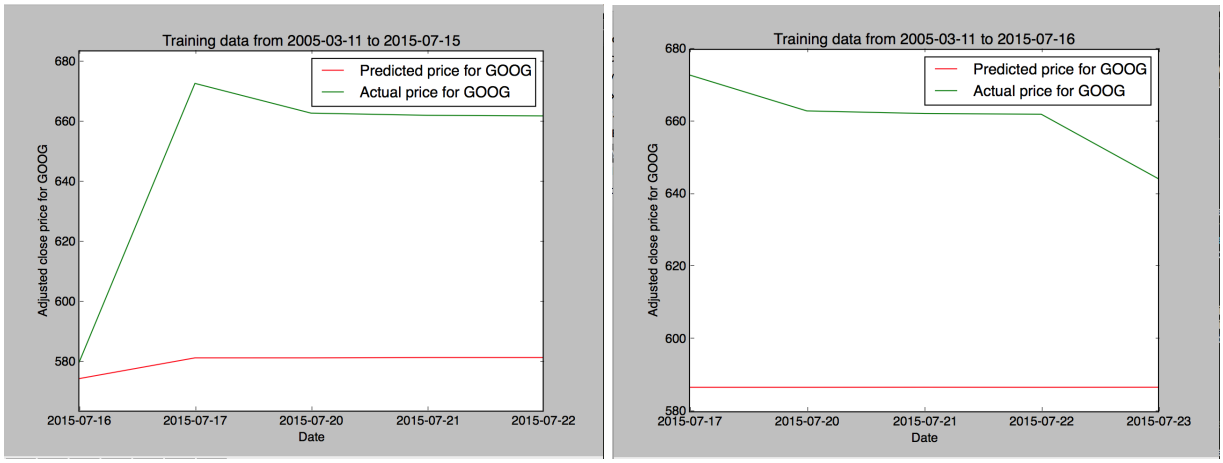
```
$ python main.py
Developer mode on
Training for FB...
Average error range for FB: $0.6151
This is the result for FB:
Training data are from 2005-03-11 to 2015-06-19
Predicted adjusted close value for 2015-06-22 is 83.9476
Actual adjusted close value for 2015-06-22 is 84.7400
Predicted adjusted close value for 2015-06-23 is 84.4460
Actual adjusted close value for 2015-06-23 is 87.8800
Predicted adjusted close value for 2015-06-24 is 84.4354
Actual adjusted close value for 2015-06-24 is 88.8600
Predicted adjusted close value for 2015-06-25 is 84.5438
Actual adjusted close value for 2015-06-25 is 87.9800
Predicted adjusted close value for 2015-06-26 is 84.5442
Actual adjusted close value for 2015-06-26 is 88.0100

Training for MSFT...
Average error range for MSFT: $0.3393
This is the result for MSFT:
Training data are from 2005-03-11 to 2015-06-19
Predicted adjusted close value for 2015-06-22 is 44.1841
Actual adjusted close value for 2015-06-22 is 44.7012
Predicted adjusted close value for 2015-06-23 is 43.9706
Actual adjusted close value for 2015-06-23 is 44.3918
Predicted adjusted close value for 2015-06-24 is 43.8441
Actual adjusted close value for 2015-06-24 is 44.1307
Predicted adjusted close value for 2015-06-25 is 43.9684
Actual adjusted close value for 2015-06-25 is 44.1404
Predicted adjusted close value for 2015-06-26 is 43.8122
Actual adjusted close value for 2015-06-26 is 43.7633
```

The error ranges for Facebook and Microsoft are less than one percent, and all the predictions for the next day(2015-06-22) are within 2 percent variation which met our benchmark. We can also see that the predictions for next week are consistent despite the fact that Facebook had a rapid growth during that week. Since this indicator mainly predicts the adjusted close price for the next day, our predictions are accurate enough to indicate the adjusted close price.

Conclusion

Free-Form Visualization

Since the predictions are all based on the past data without outliers, our indicator will fail to predict stocks that act in a completely new way with significant outliers. The following plot shows the remarkable growth for Google during July 2015.



We can clearly see the dramatic growth between July 16 and July 17. Since all the data from the past are under $600 per stock, the indicator will assume the highest closest price of all time is about $581. Thus, our predictions for July 17 and beyond are completely irrelevant due to the significant growth. Although it rarely happens, it could be crucial for some investors since the indicator could give them a terrible choice.

Reflection

Here are the tasks for the entire process:

1. Query the proper datasets from Quandl.
2. Create plots and do feature selection.
3. Setup benchmarks and learn about basic stock properties.
4. Train and test the datasets using various algorithms and parameters.
5. Figure what aspect of the features is important and preprocess the data.
6. Test the program and make sure it is robust.
7. Create user interface with various options.

The most challenging tasks are 4 and 5. I had a difficult time balancing the optimal algorithm and parameters with reasonable time complexity and results. Furthermore, since every stock has different aspects, figuring out what data are important is very challenging.

The interesting aspect of this project is the final model could make decent predictions with only free data from Quandl. Although the current final model is not enough to supervise investors, it has a lot of potentials with paid APIs because paid APIs provide detailed data. Thus, the final model should be used for suggestions which are useful for new investors.

Improvement

To improve the final model, using a more advanced algorithm or technique will not significantly improve the predictions because there are limitations with free data. Since the stock price is changed when people traded, this project needs to know more information about how people are trading.

Current news influences how people are trading since the general public uses the current news as their primary resource for how well the company is doing. Then, trading transactions give information about how people are reacting to the current news. Because of that, this project needs more detailed data that contain detailed trading transactions and current news to improve the predictions significantly.