Machine Learning Engineer Nanodegree

Capstone Project

Jonathan Sullivan

November 28th, 2016

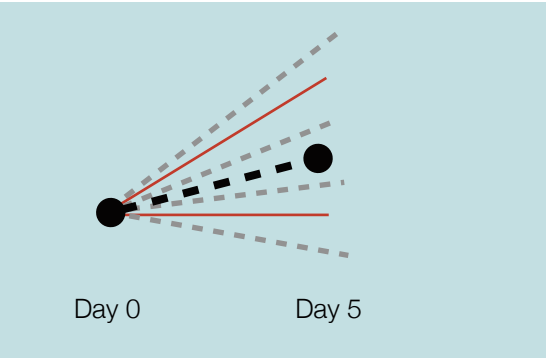
Definition

Project Overview

In this project we will use machine learning techniques and historical stock data to predict stock price for technology stocks. We will take past price data imported from the Quandl API, such as opening, closing, high, and low prices, and compute key indicators. After preprocessing this data we will feed it to our algorithm to train it to make predictions about future prices. The more innovative tools that hedge funds could use to help them make decisions in the market will not only decrease volatility and better stabilize the market but lead to higher returns into the future. Highly-reputable firms that use machine learning techniques generate very high returns for themselves and their customers. An example of some of these companies include Two Sigma Investments, D. E. Shaw (company), Renaissance Technologies, and Hudson River Trading. Matter of fact they offer some of the highest compensation packages available in the market to machine learning engineers who can identify and execute on profitable trades. One example that a Huffington Post blogger discovered is the Hathaway Effect (http://www.huffingtonpost.com/dan-mirvish/the-hathaway-effect-how-a_b_830041.html). In this article the author explains how every time Anne Hathaway's name is mentioned positively in the news the stock for Warren Buffett's Company Berkshire-Hathaway goes up.

Problem Statement

As more people use the same strategies in the market the less effective they become. Similar to a new knife if one person uses it on a day to day basis it will stay sharp longer than 100 or a 1000 people using it on a day to day basis. The problem here is we need a new knife. The proposed solution to this problem is to build a stock price predictor using supervised regression learning that takes daily trading data and statistics over a certain date range as input and outputs projected estimates of adjusted close for given query dates. This estimate should include not only a price but also the uncertainty. This estimate of the price can best be accomplished through an ensemble of regression learners (https://classroom.udacity.com/courses/ud501/lessons/4802710867/concepts/48002703630923#). The input to the regression learners will be principle component of original data and calculated normalized indicator such as simple rolling average, and Bollinger bands ®. I we measure uncertainty for the predicted prices for a n-day forecast would be the mean of the standard deviation of the all possible consecutive n-day historical prices in the training data. By looking at uncertainty this way we are saying the probability of company's instantaneous volatility being X can be modeled as the a Gaussian distribution do to the central limit theorem. When we combine this with the fact that the markets are normally less volatile at any point than they were in the past we can use this method to create a conservative range of most probable adjusted close ranges with a certain confidence. For example, if we do a 5 day forecast we look at every 5 day period in our training data. We measure the volatility of each on of these periods. We take the mean and standard deviation of these measures and use them to construct our confidence intervals.



The probability of true price being within inner dotted grey lines is .16. The probability of true price is within solid red lines is .5. The probability of true price is within outer dotted grey lines is 0.84 .

Metrics

I will evaluate the performance of our algorithm by measuring the amount of error produced by the algorithm. This will ensure that we know how far we are from the true price. We could sum these errors up but if we are adding negative an positive error then we could be misled to think our prediction are more accurate than they are. Instead we could square the error term to make sure all of our error terms are positive. However we are still left with the dilemma that by taking the sum of the squared error. the problem is that this measure will naturally increase as the amount of data added increases. Instead we can divide this measure of error by the total amount of data points. We call this the mean of squared error, MSE.
Mathematically we can define the MSE by as

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2$$

where n is the amount of datapoint, and Y-hat is the estimate price and Y is the true price for the 1st though ith point. We choose use MSE instead of a measure of correlation like R^2 is because we are less concerned about how the data trends or how closely it trends with a line and more concerned with how far the true price is from the predicted price, the error. To ensure that the regressor is a good fit for our data I will use the mean squared error. To prevent overfitting we will focus on improving the mean squared error value within our testing set. We will also employ roll-forward-split validation and grid search to tune our parameters and hyper parameters and improve performance, minimize the mean squared error.

Analysis

Data Exploration

In this project we are using the Quandl API to gain access to the datasets we need. We will first start by setting up a function called `get_data` to query the Quandl API to get data for specific symbols and dates. Next we will use this function to get data for the the following companies:

- Apple (AAPL)
- Alphabet (GOOG)
- Microsoft (MSFT)

- Amazon (AMZN)
- Facebook (FB)
- Exxon Mobile (XOM)

I decided to use the today's date, November 28, 2016, with a 24 month look-back period plus the n-days necessary to compute the n Simple moving average and n- Bollinger Values ®. We also set a p-day value, where p is the number of days in a p-day forecast. After inspecting this data we see that we have the following normalized information in this order:

- Open
- High
- Low
- Close
- Volume
- Ex-Dividend
- Split Ratio
- Adj. Open
- Adj. High
- Adj. Low
- Adj. Close
- Adj. Volume

Next we will front-fill the missing values. After all the data have been front filled we will back-fill all missing values. Next we will drop the following features in data because they are represent more truly in the other features:

- Open: the
- High
- Low
- Close
- Volume
- Ex-Dividend
- Split Ratio

The adjusted price values differ form the unadjusted values by making the historical share amounts invariant. This means when a stock split occurs we simply pretend it didn't. By using adjusted price values only, we don't have to consider ex-dividends and split ratios. Next we will compute the n-simple moving average, p-day rolling standard deviation, and Bollinger values ® for the prices in the data. We chop off the first n-days of the data so that we can compute values just over the look back period. We then segment the data into segments useful for computing:

Price features Dataframe

- Adj. Open (minus last day)
- Adj. Close (minus last day)

Adj. High (minus last day)
Adj. Low (minus last day)
Adj. Open n-day SMA (minus last day)
Adj. Close n-day SMA (minus last day)
Adj. High n-day SMA (minus last day)
Adj. Low n-day SMA (minus last day)
Adj. Open Bollinger (minus last day)
Adj. Close Bollinger (minus last day)
Adj. High Bollinger (minus last day)
Adj. Low Bollinger (minus last day)
Adj. Open n-day Momentum (minus last day)
Adj. Close n-day Momentum (minus last day)
Adj. High n-day Momentum (minus last day)
Adj. Low n-day Momentum (minus last day)
Adj. Volume (minus last day)

Price Label Dataframe

- Adj. Close (minus first day)

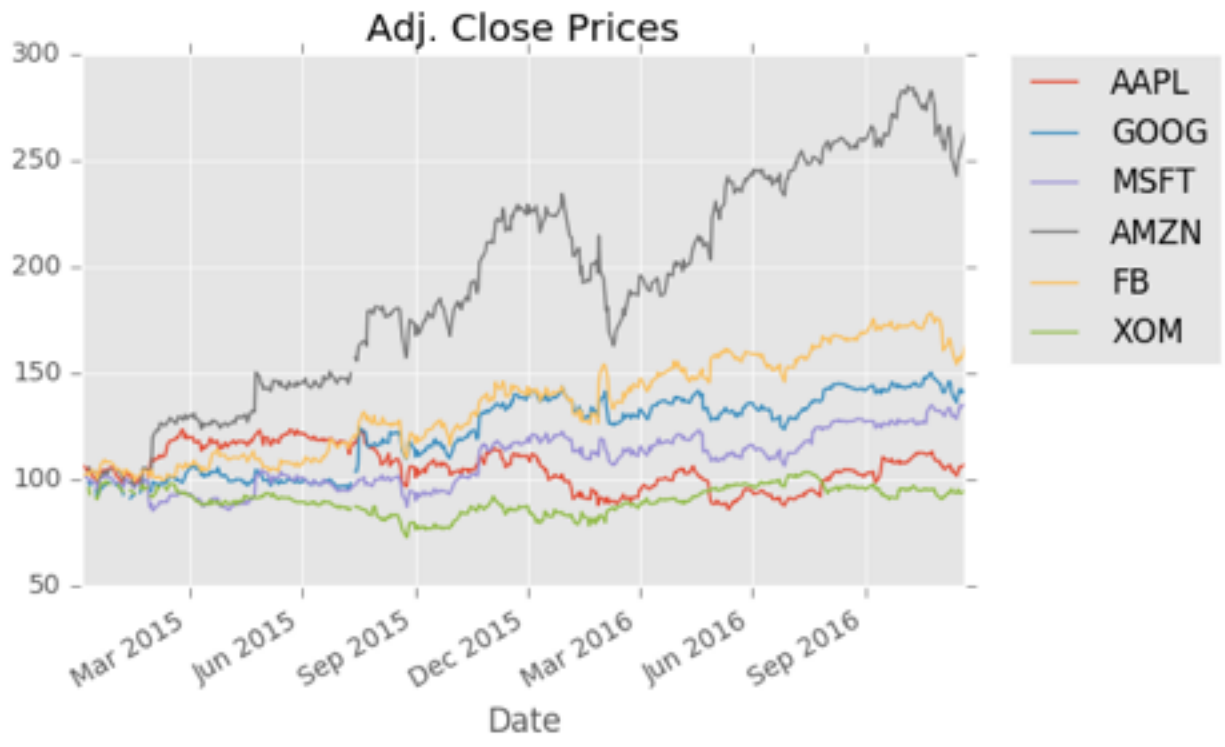
Certainty Dictionary

- Adj. Close p-day rolling standard deviation's mean
- Adj. Close p-day rolling standard deviation's standard deviation

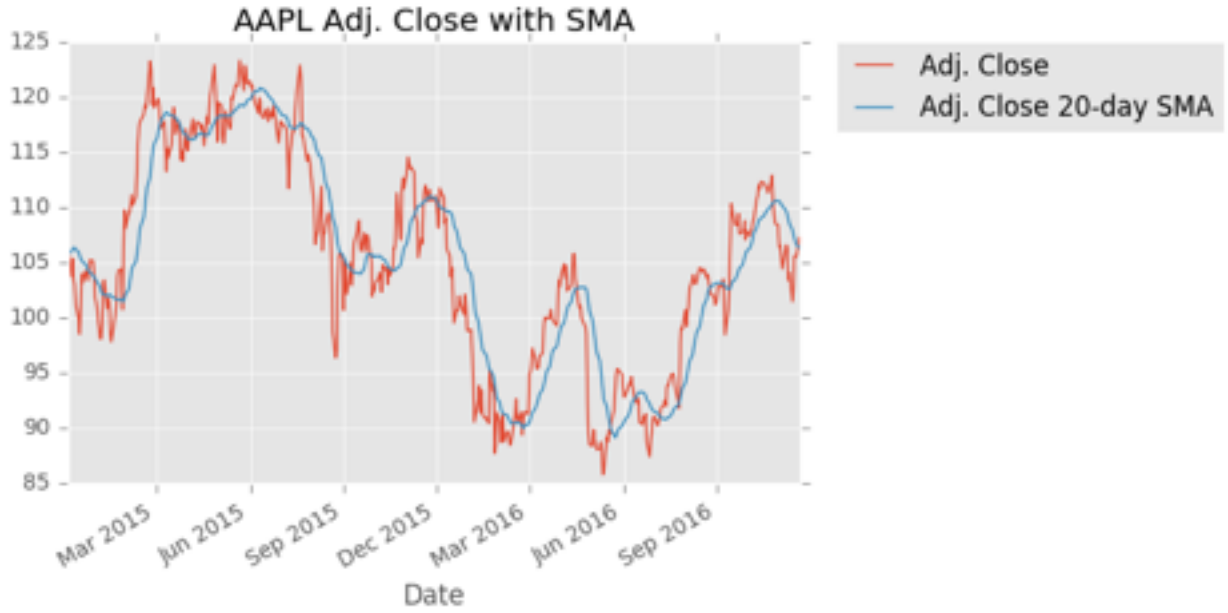
In the above list open and close represents the opening price and closing price for a stock on a given day, high and low represents the highest and lowest price for a stock traded for on a given day and the Adj. Close p-day rolling standard deviation's mean is the rolling standard deviation is the mean all rolling standard deviation of length p-days. Now that the data has been segmented, and preprocessed we can now start to explore the feature and think about Principle Components. The Bollinger values ® are 2 times the rolling standard deviation. To form Bollinger bands ® we find the simple moving average, s, and the Bollinger value, b. We then can say the upper Bollinger band ® is s + b and the lower Bollinger band ® is s - b.

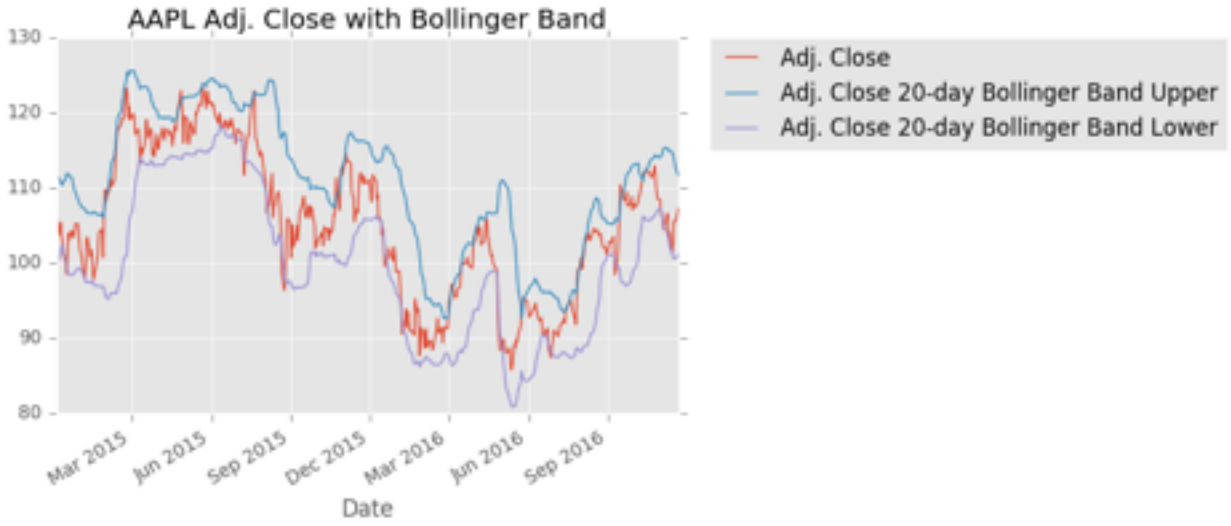
Exploratory Visualization

Below we see the normalized stock prices for the equities in question. As we can see, compared to its competitors AMZN has increased it perceived equity the quickest over the past 24 months, and that there was little change in XOM prices. Going long would have been a good investment choice for Amazons stock in Q1 of 2015.



We can explore also the closing price of a particular stock with Bollinger Bands ® and the closing price of a particular stock with the simple moving average. We see here that Bollinger Bands seem to be very affective as stock indicators for trading.
Lets take a look at Apple(AAPL).





To the right we can see the uncertainty of our prediction given past values. We can take this to mean that with 16% confidence that our predicted price should not differ from the actual price by 0.49, 50% confidence that our predicted price

should not differ from the actual price by 1.42, 84% confidence that our predicted price should not differ from the actual price by 2.34, 97.5% confidence that our predicted price should not differ from the actual price by 3.27 and 99.85% confidence that our predicted price should not differ from the actual price by 4.19. Remember these values are conservative because they look at past data.

Algorithms and Techniques

To solve this problem we will implement two different algorithms in a ensemble of regression learners. An ensemble of learners is a set of multiple learning algorithms that obtain better predictive performance than its individual part working independently on its own.The first regression learner produces an instance based model and is named K- Nearest Neighbors. K-Nearest Neighbors use the k closest training examples in the feature space to predict the output, which is the k closest training examples average value. The second regression learner produces an parameterized model and is named Polynomial Regression. Polynomial regression is nothing but an extension of linear regression made by constructing polynomial features. Instead of finding the minimum mean of squares for y = m * x + b, we find the minimum mean of squares for y = m_1 * x + m_2 * x^2 + m_3 * x^3 + m_i * x^i … + b when using a polynomial of degree i.

Our input to these to regression learners contain two components, features and labels. Our feature set is the 3 principle components that we have computed above and our labels are our dataframe of Adj. Close Values. Principle components are derived by Principle Component Analysis. “Principal component analysis (PCA) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components. The number of principal components is less than or equal to the number of original variables. This transformation is defined in such a way that the first principal component has the largest possible variance (that is, accounts for as much of the variability in the data as possible), and each succeeding component in turn has the highest variance possible under the constraint that it is orthogonal to the preceding components. ”(https://en.wikipedia.org/wiki/Principal_component_analysis) By using 2 different types of supervised regression learners we concede that neither one of these learner are ‘perfect’ for solving this problems. Instead they both have relative advantages that could prove useful and disadvantages that could prove obstructive. By implementing a solution

that takes in two separate guesses from our supervised regression learners, we can amplify the advantages these learners offer and condense their disadvantages. In fact, we are taking mean prediction for these two learners.

GridSearch is implemented so that we can tune the parameter and hyper parameters of our regression learner.
As far as cross-validation, a method is used where we don't “peek into the future”. Let say we want to predict the stock of Apple’s price 5-days from now using 20 day rolling statistics with a loopback period of 3 months and a study period of 3 years. The data gathered and pre-processed will contain 3 years of principle component data. We take the first 3 months (90 days) of this data as a feature and the 95th day adjusted closing price as our label. We then move forward one day in time to produce our next feature/label pair. We continue to iterate until we have exhausted the list. Once the list has been exhausted our model will have been trained. At this point, we simply ask it to predict the Adj. Closing Price 5 days from now.

Benchmark

We use market performance as the benchmark. Even though we are unable to peek into Blackstone hedge fund APIs and view their models, we objectively measured the model that we created and compared it to the Naive Prediction of Stock Prices. “The naive prediction asserts todays stock price as the best estimate of tomorrows price. This is a direct consequence of the random walk hypothesis It is always a good idea to measure the goodness of a predictor in relation to this trivial predictor” (http://e-m-h.org/HeHo98.pdf)

Methodology

Preprocessing

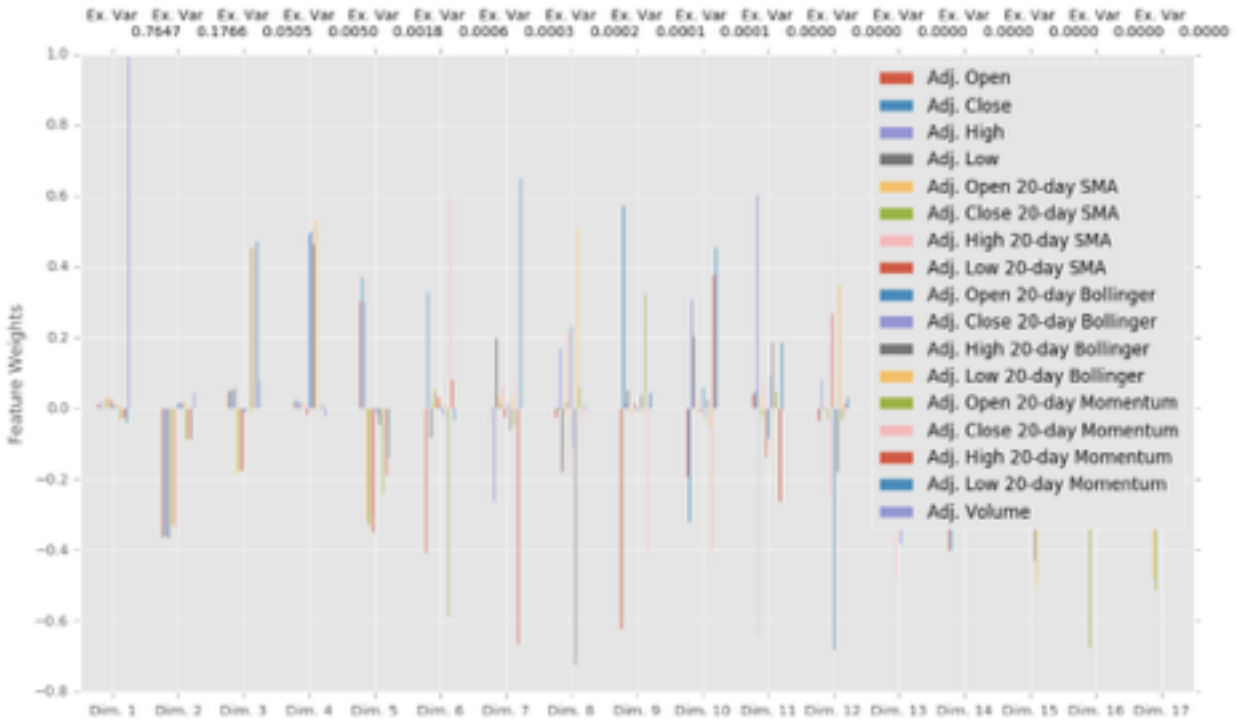
When it comes to normalization of the price data, Quandl has handled this for us. Also we have computed key indicator for all the price feature values. At this point in the process we have data with the following features:

Price features Dataframe

- Adj. Open (minus last day)
- Adj. Close (minus last day)
- Adj. High (minus last day)
- Adj. Low (minus last day)
- Adj. Open n-day SMA (minus last day)
- Adj. Close n-day SMA (minus last day)
- Adj. High n-day SMA (minus last day)
- Adj. Low n-day SMA (minus last day)
- Adj. Open Bollinger (minus last day)
- Adj. Close Bollinger (minus last day)
- Adj. High Bollinger (minus last day)
- Adj. Low Bollinger (minus last day)
- Adj. Open n-day Momentum (minus last day)
- Adj. Close n-day Momentum (minus last day)
- Adj. High n-day Momentum (minus last day)
- Adj. Low n-day Momentum (minus last day)
- Adj. Volume (minus last day)

Due the curse of dimensionality, so many features would require a huge amount of data. Of course we could loopback further in time at the price of the stock we are looking at but data from the market 10 years ago is irrelevant to market behavior to day. Instead we will try and see if we can reduce the dimensionality of the feature space. This can be accomplish by seeing which features are more highly correlated and removing less correlated features. This method i don't feel would be affective for time series data. Instead we used principal component analysis.

Feature Weights with Explained Variance of Principle Components for AAPL



HERE WE CAN LOOK AT THE FEATURE WEIGHT OF EACH PRINCIPAL COMPONENTS FOR APPL

We can see from the AAPL example that the first 3 components explains over 99% of the variance in the data. In fact, the other high-valued tech stocks we are looking at also follow this pattern. When it comes to feature reduction 3 features would be optimal for evading the curse of dimensionality.

Implementation

We first start by importing all needed libraries.

Cross-Validation

- sklearn.model_selection.TimeSeriesSplit
- sklearn.model_selection.GridSearchCV

Metrics

- sklearn.metrics.mean_squared_error

Regressors

- from sklearn.preprocessing import PolynomialFeatures
- from sklearn.linear_model import LinearRegression
- from sklearn.pipeline import Pipeline
- sklearn.neighbors.KNeighborsRegressor

Next we write a function handles our bagging. This function trains the K Nearest Neighbors Regressor and the Polynomial Regressor using a exhaustive grid search. We then write a function to take the predictions of this ensemble of learners to compute our final prediction value. We then write a wrapper function that allows us to abstract the underlying process into a training function. This training function returns our two models fit to our training data. The last function we write is the predict function. This function takes in the output of our training function and our test features. This function return a vector of stock predictions.

We split our data using a roll-forward-split algorithm, which is appropriate for time-series data. This splits gives us multiple training and test sets for all of out tickers. We then store these sets and feed them one by one into our training function and store the created model for each stock. Once we have the model for a given training set on a given equity, we use our testing features to predict the price for the corresponding given testing set. We also store these predictions.

Next we record the mean squared error value for set of predictions when compared to the true prices. We also measure how many prediction were within 1 p-day mean standard deviation of the predicted price. This way we not only have a measure of precision but accuracy also.

Refinement

To tune the parameter and hyper parameter in the algorithms used in the regression, I used and exhaustive grid search. In fact, it was too exhaustive. After taking an hour to fit a K nearest neighbor regressor, I decided that we should use to reason and intuition to reduce the number parameter being searched. Instead we reduced the search space kNN as follows:
metric: (‘euclidean', ‘manhattan', ‘minkowski') -> (‘euclidean', ‘manhattan’)
’n_neighbors’: (range(X_train)) -> (range(len(X_train)/2)[(1*len(X_train)/4):(3*len(X_train)/4)])
'algorithm':('auto', 'ball_tree', ‘kd_tree’, ‘brute’) -> (‘auto', 'ball_tree', ‘kd_tree')

The Minkowski distance can be ignored because it can be considered as a generalization of both Euclidean and Manhattan metrics. The number of neighbor near the minimum and maximum will be know as useful because it will either overfit or underfoot the data. So i choose to use the middle half instead. Using brute force algorithms always are longer and less efficient than intelligent algorithms. This led to faster training times and faster overall time.

Results

Model Evaluation and Validation

Below is the performance of our algorithm for each company that we are studying along with performance of the Naive Prediction of Stock Prices.

AAPL Performance				
	Our Model		Benchmark Model	
Future Dates	**Mean of Square error**	**% within one Std. Dev**	**Mean of Square error**	**% within one Std. Dev**
1-Day	2.35	42.5	1.87	53.0
5-Day	3.19	54.0	8.24	49.71
10-Day	6.47	48.21	14.37	24.83
30-Day	6.47	76.92	32.21	0

GOOG Performance				
	Our Model		Benchmark Model	
Future Dates	Mean of Square error	% within one Std. Dev	Mean of Square error	% within one Std. Dev
1-Day	1.09	44.5	1.15	51.0
5-Day	2.01	52.5	4.39	49.14
10-Day	3.05	58.97	6.71	42.76
30-Day	3.05	81.03	23.21	28

MSFT Performance				
	Our Model		Benchmark Model	
Future Dates	Mean of Square error	% within one Std. Dev	Mean of Square error	% within one Std. Dev
1-Day	2.83	38.5	2.26	54.0
5-Day	4.40	45	7.00	50.29
10-Day	6.42	45.64	10.38	35.17
30-Day	6.42	62.05	32.41	30

AMZN Performance				
	Our Model		Benchmark Model	
Future Dates	Mean of Square error	% within one Std. Dev	Mean of Square error	% within one Std. Dev
1-Day	12.27	9.5	7.39	54.0
5-Day	18.81	12.5	18.95	45.71
10-Day	23.61	30.77	26.21	33.79
30-Day	23.61	54.36	65.66	0

FB Performance				
	Our Model		Benchmark Model	
Future Dates	**Mean of Square error**	**% within one Std. Dev**	**Mean of Square error**	**% within one Std. Dev**
1-Day	6.56	24.0	4.39	50.0
5-Day	9.67	27.5	11.01	50.86
10-Day	12.4	40.0	16.79	42.07
30-Day	12.4	56.41	35.14	12

XOM Performance				
	Our Model		Benchmark Model	
Future Dates	**Mean of Square error**	**% within one Std. Dev**	**Mean of Square error**	**% within one Std. Dev**
1-Day	7.86	26.5	4.54	55.0
5-Day	8.90	30.5	7.00	50.29
10-Day	42.56	42.56	10.13	33.79
30-Day	8.19	57.44	10.03	36

Percent within one mean standard deviation of prediction for XOM is 46.6666666667%
The parameter for our kNN Regressor for the equity AAPL are {'n_neighbors': 93, 'n_jobs': 1, 'algorithm': 'auto', 'metric': 'euclidean', 'metric_params': None, 'p': 2, 'weights': 'distance', 'leaf_size': 30}
This model seem to be very robust for the case of long forecast. We are able to predict the price of of long forecast on stock most of the time within a reasonable range.

Using grid search I optimize the degree of the polynomial in our parameterized model. What this does is test out different values of polynomial degree in different ranges and cross validates them to make sure over citing isn't produced. If the degree is too small then the mean squared error would be high and the model will underfit the data but if the degree is too large the data will be overfit however the mean squared error would drop. This is similar with the k variable in the kNN algorithm, if k is too large the mean squared error will be high and the data will be

underfit. However if the value of k is too small then the mean squared error will be low however the model will overfit the data

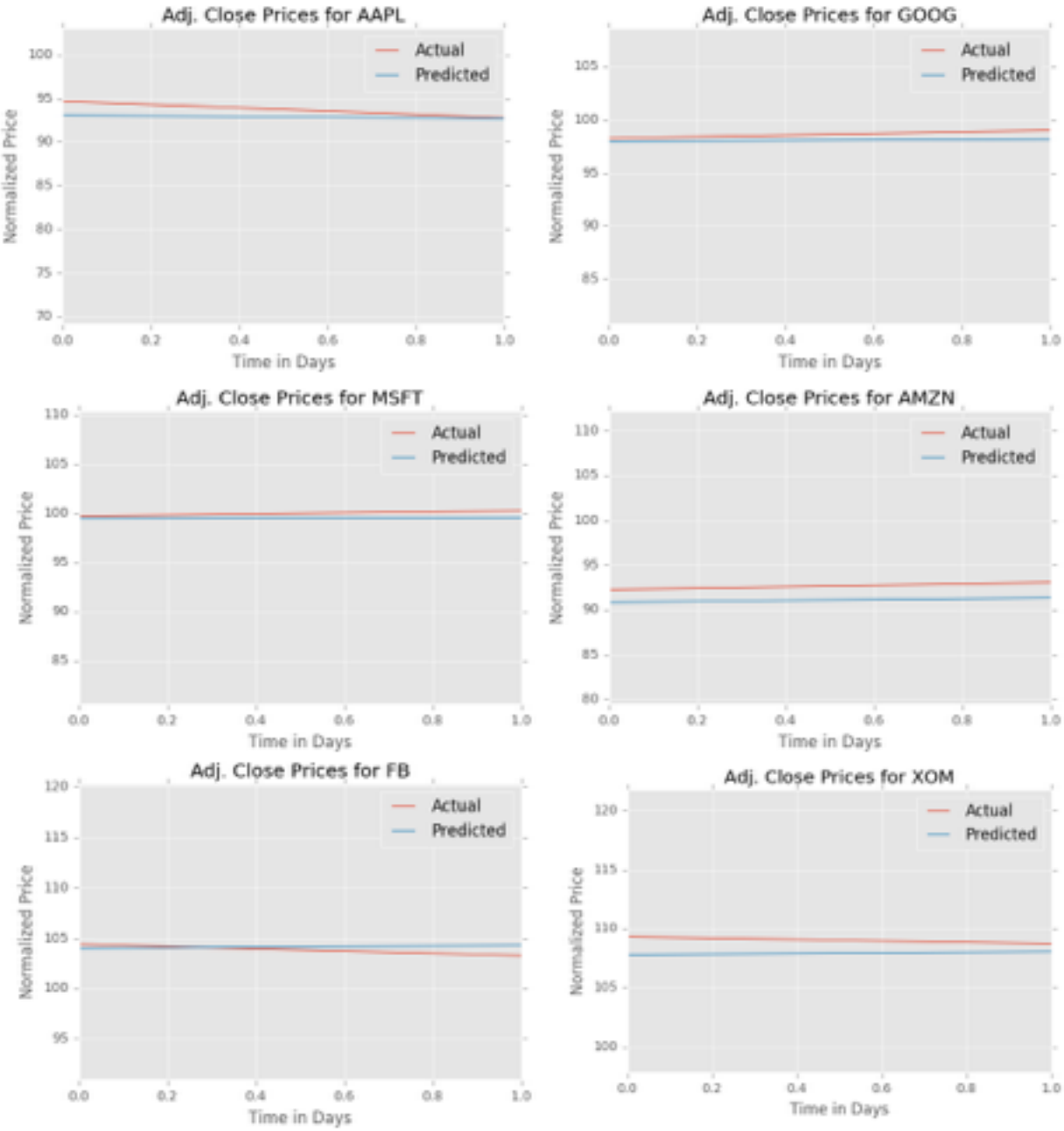
Justification

Surprisingly on small forecast prediction our algorithm was out performed by the benchmark. However the longer the forecast period got the better our algorithm performed compared to our benchmark. Using this predictor takes out some of the inherent risk of investing in the market. Applying this algorithm to gain information about different diverse array of stocks will lead to positive increases in return in the market.

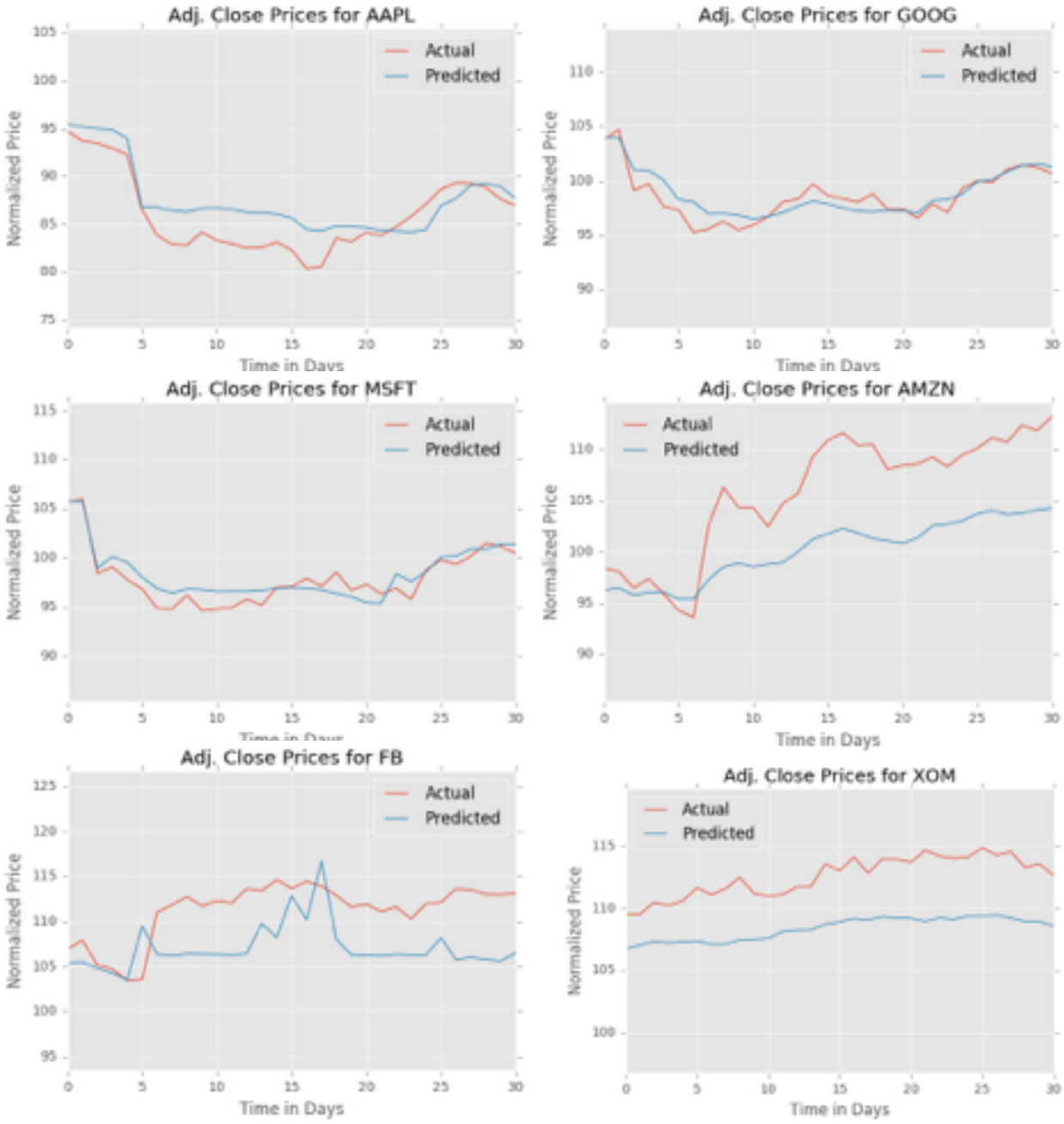
Conclusion

Reflection

When comparing the original process proposed to the actual implementation, we notice a few differences. However most of the processes were the same. We used Quandl python API to import financial data and matplotlib to visualize it. We then preprocessed this data by looking at its principle components, by segmenting it into features and labels, and computing stock price indicators. We then computed the average standard deviation in price for the forecast period. We then used a roll forward split to separate the data into training and testing sets. Next we assembled our learner and ran our data through a grid search to optimize the model parameter and hyper parameters. Then used the mean of the squared error and a custom metric based on if the predicted price is in an acceptable range of certainty. We then reported those results. I honestly was surprised how the algorithms reacted differently to different equities.



Above are some visualizations that show the predicted versus actual prices for a 1-day forecast



Above are some visualizations that show the predicted versus actual prices for a 30-day forecast

Improvement

This algorithm seem to perform well for stocks like AAPL, XOM, but not so well for AMZN as we saw previously. I think it would be fruitful to look at different algorithm for stock prices that grow relatively fast. Also I found it odd that predicted stock prices with respect to our measure of variability became more accurate the further in time we predicted. These algorithm work best on stock whose volatility is low. Even though future events seem unpredictable with machine learning we see that everything even future stock prices follows a recognizable patterns.