Machine Learning Engineer Nanodegree

Capstone Project

Dillon Pietsch

July 24, 2017

I. Definition

Project Overview

Overview

This project was about using supervised algorithms and data preprocessing and manipulation to predict the price of a stock x days into the future. Changing the way the data was originally given allows the algorithms to make better decisions and see more correlations between certain features in the data. Also, the algorithms wouldn't work in some instances without certain things (like non-existent values AKA NaN) being taken out and replaced with proper values. This project looks into the Opening Price ('Open'), highest price intra-day ('High'), lowest prices intra-day ('Low'), the 20 day rolling average ('20 Day Rolling Average' which looks at the stocks momentum), and the 50 day rolling average ('50 Day Rolling Average') in order to predict the Adjusted Closing price, and not the regular closing price due to company actions like splitting a stock which would change it's value without an actual increase/decrease in the companies value itself.

Domain

For as long as computers have been viable and powerful enough, people have sought to predict stock prices. Many hedge funds use machine learning in their box of tools to help 'predict' future stock prices. This project has very few features, but other models would take into account news about the company and many more statistical methods to determine the direction of the company. However, not even the best algorithms are able to predict the prices 100% accurately, so there are even more unknown factors to be found. But the field continues to grow and sees more investment year by year into machine learning techniques.

Problem Statement

The problem that is trying to be solved is how does one use supervised learning algorithms to predict where a stock price will be into the future.

There are many unknowns in a company that prevent a better understanding of the company which would help to know more about the stock's future. Some of these unknown pieces of information are:

The amount of revenue the company made in a given quarter (until they release that information to the public).

The news and subsequently the public opinion of the company in the future, which can affect the stock price.

Changing of C-Level executives that might change the outlook of the company for better or worse.

There is yet to be a way to use past trends in the stock to find the price of the stock in the future, if it's even possible.

Solution to problem

Through this project I have tried to solve the problem of predicting stock prices by using supervised learning algorithms. I chose supervised learning algorithms because the data is clearly labeled and it is known what is being predicted - that being a stock price into the future. Through the pandas-datareader module I retrieve multiple features of information as discussed in the overview. On top of that I added extra features to show the progress of the stock and give a clearer picture of where the stock pirce is heading in the future. The data is normalized on a scale of 0 to 1 to minimize large volumes of certain features and not skew the algorithms predictions as they might be subject too. For example, the volume feature is the number of trades of a given stock in a day, and that can range widely so normalizing the values are a must. Once the data is cleaned and normalized the supervised learning algorithms are able to be trained and eventually make predictions on future stock prices. The close column is dropped from the dataset because it would help the algorithms 'cheat' by knowing what the close is and only the Adjusted Close ('Adj Close') is the true indicator of the price.

Metrics

The metric I have decided to use to measure the accuracy for this problem is R^2. Since we are predicting stock prices (regression values), R^2 is useful because it shows how well the line is fitting for our given data. R^2 ranges from 0.0 to 1.0, with 0.0 being the lowest score. The score is calculated by subtracting the quotient of the sum of squared errors by the total sum of squares.

II. Analysis

Data Exploration

Data description (from Tesla dataset)

Descriptor	Open	High	Low	Adj Close	Volume
count	1828.000000	1828.000000	1828.0000000	1828.000000	1.828000e+03
mean	150.619081	153.125684	147.929568	150.585093	4.463941e+06
std	106.364284	107.774950	104.754838	106.306310	4.245858e+06
min	17.799999	17.900000	17.389999	17.600000	1.185000e+05
25%	31.297500	32.000000	30.665000	31.355001	1.283775e+06
50%	185.159996	188.889999	182.069999	185.295006	3.501800e+06
75%	230.790001	234.902504	227.182499	230.524998	5.978575e+06
max	386.690002	389.609985	379.350006	385.000000	3.716390e+07

There are no real outliers in the data. The values that change widely from one value to another are still relevant to the data, and there is some other source of information that is not being taken into account in this project (ex: news about the company). A standard deviation of about 100 on all of the features besides 'Volume' shows the little variance there is, considering that the stock changes from around a min (on all features besides 'Volume') of ~17.7 to a max of ~383. Volume, however, does have higher variance, as shown by its standard deviation being 4245858.0. Again, while there are these outliers, they are still a part of the data, and help to make a prediction. But, we're not taking into account external factors that cause these swings. I didn't do anything to outliers in this project because they are still relevant to determining the stock prices on any given day.

Snippet of Tesla dataset

Open	High	Low	Adj Close	Volume
355.559998	357.149994	348.200012	351.809998	5061800
352.690002	354.549988	344.339996	345.100006	4930400
349.880005	349.950012	336.250000	337.019989	5747300
338.799988	342.799988	336.160004	337.339996	4491700
336.700012	337.500000	323.559998	325.839996	8576800

Feature explanation

Open: The opening price of the stock.

High: The high price of the stock during the trading day.

Low: The low price of the stock during the trading day.

Adj. Close: Stands for Adjusted close. This is the closing price of the stock at the end of the trading day, while accounting for stock splits.

Volume: The amount of shares traded during the trading day.

Exploratory Visualization

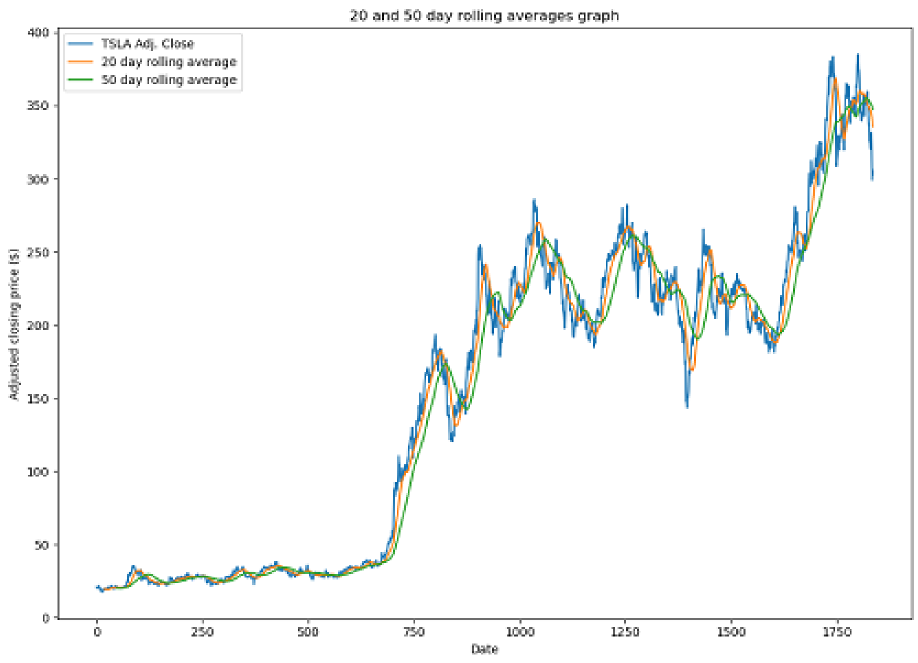

Figure 1: Rolling averages

This is a chart of the 'TSLA' 'Adj. Close' feature (in blue), and the 20 and 50 day rolling averages in orange and green, respectively. Rolling averages show the average of the stock over a given amount of days. This helps to show the movement of the stock in a greater sense, rather than just a day-to-day change.

Algorithms and Techniques

I am using five supervised learning algorithms for this project:

k-nearest Neighbors (Regressor): I think KNN is a good algorithm for this project because there is lots of flexibility in the hyper-parameters. This modularity of the algorithm provides the ability to get a better regression score for the algorithm. KNN chooses the result based on the similar data around it. Like Random forest regressor, this average of results leads to a greater accuracy in the results. However, the amount of customization that is provided in the algorithm leaves it open to overfitting. KNN uses different algorithm such as 'ball_tree', 'kd_tree', and 'brute' (brute force) to compute the nearest neighbors (which leads to the algorithm result). I like this algorithm because combined with GridSearchCV, it provides the ability to fit well onto the stock market data, without taking into consideration all of the external data that it is missing.

Decision Tree Regressor: A decision tree regressor is helpful in this project because they can learn easily off of most data without preprocessing. I like decision trees because their process of getting to the answer is easy to understand, and can be traced through, where other algorithms have more factors and algorithms nested inside of it that make it harder to eaisly understand the algorithm. This simplicity, however, leads to a problem with overfitting because the higher the score, the more decision trees are nested on top of each other, leading to different predictions in minute changes in input. By using GridSearch, I believe that the DTR algorithm will have good scores since it will break down the data's patterns, but this will probably lead to overfitting.

Support Vector Regressor: The algorithm being used is an SVR, which is a support vector machine designed to output a regression output instead of a classification output. It is better at handling output that has various factors and is more complex, which is better suited for a stock price that has a higher degree of complexity than the boston housing prices project that I did earlier in the term. Unlike an SVM, there is no ability to perform a kernal trick, since the kernel for the SVR is non-linear because it deals with regression. This algorithm fits this problem well because it will be able to pick up on complex patterns in the data. The use of GridSearchCV will help mitigate most overfitting in the project.

Gradient Boosting Regressor: The gradient boosting regressor is an effective algorithm for regression problems such as this one. The reason I chose it is because it's an easy-to-apply algorithm that doesn't require much tweaking to fit the data. Like DTR, the comes at the cost of possibly not 'seeing' the entire picture, and missing out on patterns in the data that perhaps the SVR would pick up on. To provide a bit more complexity, I tuned the parameters with GridSearchCV and tried to make the algorithm more complex. According to the documentation, this algorithm is good at handling outliers (via a "robust loss function"), which helps considering how volatile the 'Volume' feature is.

Random Forest Regressor: A RFR is a regression algorithm that acutally uses multiple classification decision trees on random sub-sections of the data, and averages it out to provide. The averaging of the results helps to prevent overfitting and provides an accurate result. I like this algorithm because it can be compared to the decision tree regressor and have the ability to compare the performance by the two. I expect this algorithm to perform better because it uses an average of multiple decision trees.

I also use GridSearchCV to chose the best parameters for the algorithms. This helps to create a better fitted model much more easily.

I am using TimeSeriesSplit due to the data being a time series format. This function is necessary because a random shuffling of the data leads to the algorithm's being able to 'see' into the future, leading to an unfair advantage.

Benchmark

I am using the Random Forest Regressor algorithm with no modified parameters. This will show what an algorithm with no tuned parameters will predict compared to one that has the best parameters possible. This show how much of an improvement the algorithm

III. Methodology

Data Preprocessing

I have dropped the Close feature that is part of the data pulled with the program. I did this because it's just a copy of 'Adj. Close', and would give the algorithm a way to cheat and know more about the data then it should. Besides that, all the data is left the same.

Before the data is fed through the algorithms I use the StandardScaler function to normalize the values.

The data is trained on the 'Open', 'High', 'Low', and 'Volume' features in order to predict the 'Adj. Close' target values.

Implementation

All algorithm's are trained on sklearn's 'TimeSeriesSplit', due to the linear nature of the data. Without this function, algorithm's are given the answers ahead of time and know what the data is going to be like.

k-nearest Neighbors (Regressor): For kNN I use multiple parameters (done automatically by GridSearchCV) to help fit the algorithm better. The first parameter I use is n_neighbors, which is checked from a range of 1-11. Also, the weights parameter has both uniform and distance to pick from to best model the data. Along with those two parameters, the algorithm that computers these weights has either 'ball_tree', 'kd_tree', or 'brute' to choose from. Finally, lear_size has a range of 10-40, providing a wide range to better fit the data for more unseen patterns.

Decision Tree Regressor: For DTR, I have put the 'min_samples_split' and 'min_samples_leaf' arguments into GridSearchCV. These two arguments define what constitutes a decision tree leaf and the branches that are a part of it too. They both range from 1 to 5, allowing for a custom model fit to occur for the problem. It allows the tree to be optimized for the problem at hand.

Support Vector Regressor: I had two different parameters that I tested with GridSearchCV. The first one was the 'kernel' argument. The degree of the kernel (especially since there is no kernel trick available) is very important, because the data is complex and requires more degrees of freedom. I tested the 'rbf', 'linear', 'poly', and 'sigmoid' kernels. These different options allow for unknown patterns to be easily found out and fitted to with the algorithm. The other parameter was the 'C' argument, which helps in generalizing the data. While it defaults to 1, I added in a range from 1 to 11, allowing for a greater chance for the algorithm to have a trade off between overfitting and generalizing to the data.

Gradient Boosting Regressor: The loss parameter checks for the best one out of 'ls', 'lad', 'huber', and 'quantile'. It measures the loss of the algorithm, and adds complexity to the algorithm, allowing it to gain a better idea of how the data works, increasing the R^2 score. Also, to expand upon that, the max_depth parameters is checked with a range from 1-11, possibly giving a deeper complexity to the algorithm and better train/test score.

Random Forest Regressor: I placed two different types of RFR algorithm in the project. The first one is a benchmark model with no parameter tuning available to it. The other RFR algorithm has the n_estimators feature checking from a range of 5 to 15 estimators using GridSearchCV.

Refinement

Initially, as shown through the benchmark model, I just have the algorithm predict based off of the values. In order to improve these results, without spending too much time manually checking the values, I used Grid Search CV in order to use the best possible parameters for the algorithm and improve the prediction. I also didn't normalize the values at the beginning which threw off Algorithms like SVR. That's when I started to use the StandardScaler function and that improved the algorithm's prediction. Adding more parameters to the GridSearchCV also improved the models too because it gave GridSearchCV more options to create a better fit to the data. Also, I didn't normalize my values, which lowered every algorithm's output by about 0.05 (on R^2 scale), so it was important to have that in there.

KNN: I only had the n_neighbors argument tried in the beginning of my testing, resulting in scores around 0.9, but adding 'weights', 'algorithm', and 'tree_size', improved this score up to 0.99 as well.

DTR: This was the only straight forward algorithm, I gave it 'min_samples_split' and 'min_samples_leaf' with 1 to 5 range and it outputted around 0.99 score already, so I felt like it was at its max potential already, so I stopped adding parameters there.

SVR: Initially, I had no parameters in the GridSearchCV for the algorithm. This lead to results around .93, which is good. I improved the score to .99 by adding the kernal parameter and testing 'rbf', 'linear', 'poly', and 'sigmoid' kernels. I also added a the 'C' parameter with a range of 1 to 11 as to give the algorithm a greater chance to fit to the data.

GBR: Without any customized parameters, GBR scored 0.845, and adding in 'max_depth' with a range of 1 to 11 lead to a score increase to 0.93. I then added in the different 'loss' options, and that gave the algorithm a jump up to 0.99 too, which wasn't surprising considering that the loss adds complexity to the algorithm and giving it a greater chance to fit to the data.

RFR: The default algorithm already scored .99 in testing, and adding the 'n_estimators' parameter increased the score minimally. I didn't pursue increasing this algorithm further. I was surprised at how well the benchmark did. The algorithm is naturally suited towards this data.

Eventually, I changed how the training and testing data was created. Instead of using 'train_test_split', I switched over to 'TimeSeriesSplit', because 'train_test_split' ends up shuffling the data. This is an issue when it comes to predicting stocks, because if yesterday and the following day are in the training data, then the current day in the testing data has to be between those two values, and since stock prices don't change too much day by day, this gives the algorithm's easy ways to predict the stock price.

IV. Results

Model Evaluation and Validation

The final model is about as good as these algorithms are able to get. They are fitted with the best scoring parameters possible through the use of GridSearchCV. Regardless of the stock that is chosen, the model fits to about the same accuracy every time. The parameters are simplistic on all of the algorithms and that helps to create a good fit.

'TSLA' results

	R^2 Test Score	Actual Stock Price	Predicted Stock Price
Benchmark (RFR)	0.99	205.289	206.04
RFR	0.995	205.289	205.80
DTR	0.999	205.289	206.55
KNN	0.995	205.289	204.43
SVR	0.999	205.289	204.50

	Rˆ2 Test Score	Actual Stock Price	Predicted Stock Price
GBR	0.999	205.289	207.22

‘AAPL’ results

	Rˆ2 Test Score	Actual Stock Price	Predicted Stock Price
Benchmark (RFR)	0.99	98.607	98.133
RFR	0.995	98.607	95.77
DTR	0.998	98.607	97.41
KNN	0.997	98.607	97.48
SVR	0.996	98.607	97.78
GBR	0.999	98.607	97.74

‘KO’ (Coke) results

	Rˆ2 Test Score	Actual Stock Price	Predicted Stock Price
Benchmark (RFR)	0.962	41.58	41.93
RFR	0.967	41.58	42.26
DTR	0.95	41.58	42.29
KNN	0.995	41.58	41.87
SVR	0.97	41.58	42.51
GBR	0.97	41.58	41.63

It’s interesting how the models nearly ‘perfectly’ found the right match to the data. I believe this is because the features (besides Volume) are hinting at what the true answer is. There are at times, however, the that models stray away from the perfect score. I believe this is because of the volatility (explore in the visualization below) of the stocks. The Coke (‘KO’) stock seems to be more volatile than ‘TSLA’ and ‘AAPL’, so changes in the data lead to worse scores for the algorithm. I wouldn’t trust this algorithm. But, I would like to point out that the alogrithms tend to go for a lower stock price than the actual price, meaning that any trade made in anticipation with this program’s help, will get an extra profit because the stock outperforms the prediction. That seems to be one of the benefits of this algorithm. KNN ended up being the best algorithm out of the pack, which I believe is due to the number of parameters that are inputted into grid search, allowing it to fit better the data at hand. The benchmark still does amazingly well for default parameters, which is understandable because the boosting in the algorithm helps it generalize well the ‘Volume’ feature among the others, giving it a better chance of finding correlation.

Here are the new results using TimeSeriesSplit:

‘TSLA’ results

	R^2 Test Score	Actual Stock Price	Predicted Stock Price
Benchmark (RFR)	-0.27	315.05	285.07
RFR	-0.35	315.05	282.80
DTR	0.54	315.05	282.37
KNN	-0.58	315.05	277.39
SVR	0.998	315.05	318.79
GBR	0.52	315.05	281.08

‘AAPL’ results

	R^2 Test Score	Actual Stock Price	Predicted Stock Price
Benchmark (RFR)	-5.45	170.15	98.133
RFR	-6	170.15	125.56
DTR	-0.20	170.15	125.20
KNN	-5.91	170.15	127.77
SVR	0.94	170.15	163.63
GBR	-0.16	170.15	125.67

‘KO’ (Coke) results

	R^2 Test Score	Actual Stock Price	Predicted Stock Price
Benchmark (RFR)	0.273	45.71	43.54
RFR	0.256	45.71	43.62
DTR	-0.02	45.71	43.35
KNN	0.23	45.71	42.63
SVR	-0.32	45.71	42.82
GBR	-0.07	45.71	43.45

Not surprisingly, this change in how the data is handled lead to a huge drop in performance. Without the so-called ‘time machine’ feature, the algorithm’s show their lack of complexity on all sides. I’m interested in the new ways that this project can be improved from here, now that the algorithm’s are performing realistically. That being said, even with the poor R^2 scores, on the price examples, the algorithm’s are choosing the stock price to be lower than the actual amount (leading to higher profits for the user) and are pretty close to the mark anyways. There is still some weird data processing happening, as seen in the TSLA dataset having a flawless result for SVR (0.998), but having negative results for the RFR and KNN algorithms. These results are still not reliable enough to be that useful in a real life application of this program. It simply requires better algorithms to match the complexity of the problem. While the Rˆ2 scores go into the negatives for multiple algorithms (which means that the algorithm's are predicting 'worse than random'), the fact that those results are mainly below the actual prices is a good sign, and again would help out the investor. The more volatile the stock, it seems, the more that the models seem to predict less accurately. All in all, I'm happy with the results even though the algorithm's are off of the mark, because the actual values are frequently above the predicted values, which is better than the predicted values being greater than the actual values.

V. Conclusion

Free-Form Visualization

The following graphs are the volatility of each stock.

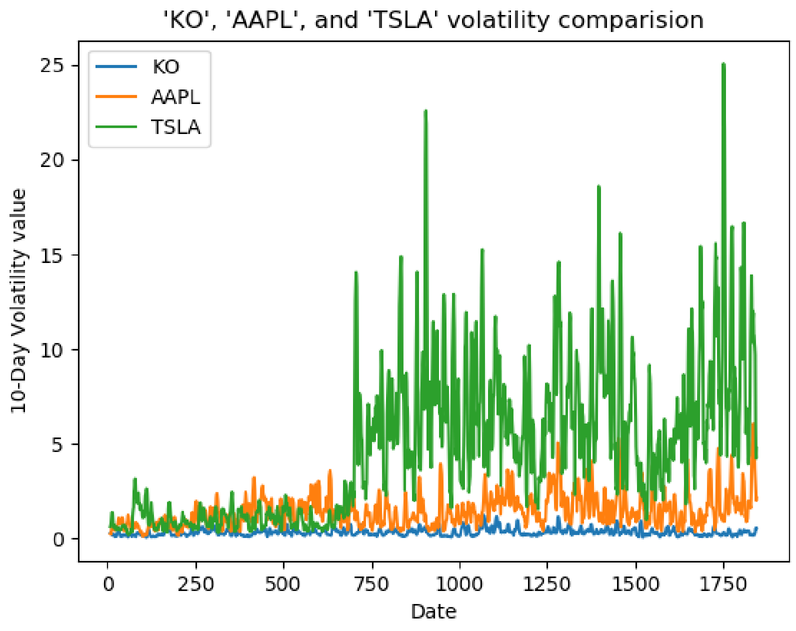

Figure 2: volatility

It's surprising that the algorithms had a harder time predicting the 'KO' price. It has very low volatility so I would've assumed that it would've easily fit to that data. Somehow, 'TSLA', with a greater volatility was very easy for the algorithms to predict.

Reflection

It was nice to use features like GridSearchCV and train_test_split to automize the process of improving my algorithm score. I had to look at the documentation for each algorithm and in that process I learned how each one of the algorithms worked and how the parameters affected the score. The most difficult part was getting caught on how it would actually work and I ended up overcomplicating the process. I initially started out without GridSearchCV and that held me back for a while. After manually entering in parameters for a while, I stumbled across this method again and applying it helped with my testing process. Another bump in the road was getting the values normalized. Other normalizers strip off the column headings and the process ended being a jumbled mess trying to re-add the column headers back on (which seemed like the solution at the time. However, I standardized the process and made the code into a class so I could clearly lay out the steps for the data to be inputted. Having the train_test_split function helped because I normalize the data right before it is inputted, and don't have to worry about the column labels or anything else that concerned me before. After I got in the groove with train_test_split, StandardScaler, and GridSearchCV, it was very easy from there on out. I made sure to remove the 'Close' feature from the data because the algorithms would have a 1:1 correlation of the prices (even though the algorithm's can see the patterns in all features besides 'Volume'). I was stuck for a while wondering if I was doing this project right due to the fact that the algorithm's did score so high on the stocks. However, there aren't many options outside of the given data to train the stocks, so it ended up being the final solution. However, I think if I were to make an actual stock predictor for a company, it would have more complex features and insight (as explained in the improvement section). I believe I did the best I could with this project and want to expand upon it and make it better when I have the time.

Improvement

A big improvement would be to use neural networks. There is a lot more customization that can come from those algorithms and that will help to better fit the data. These algorithms would be able to better take in the complexity of the data and even take in more things into consideration like news and macro-economic principles. I've basically maxed out the capability of the supervised learning algorithms, so there isn't much room for growth there.