Machine Learning Engineer Nanodegree

Capstone Project

Panagiotis Pnevmatikatos
January 10th, 2018

I. Definition

Project Overview

“Prediction is very difficult, especially about the future”

~~~ Niels Bohr ~~~

The stock market in its early form appeared in France in the 12th century and had a rise in Italy in 13th-14th centuries. Formally, the first company who issued bonds and shares to the general public was Dutch East India Company established in 1602. So, the first formal stock market was Amsterdam Stock Exchange[1]. Now stock markets exist in every developed economy. One of the biggest is the London Stock Exchange, which I will use for this project.

Prediction of stock prices was something that existed from the very beginning of stock markets. And relying only on luck was not enough for people who were trying to get profit. In 1973 Burton Malkiel[2] issued his work A Random Walk Down Wall Street. He argued that you can’t predict stock prices from the historical prices, and financial specialists, predicting the market, actually don’t help or even hurt the profit. Malkiel presented a concept of "random walk" meaning each day's deviations from the central value are random and unpredictable.

Although this work was influential, the attempts of stock predicting did not stop. Nowadays we can pick out 3 general categories of prediction methodologies: Fundamental Analysis (evaluates a company's past performance and its account credibility), Technical Analysis (determines the future price of a stock based on the trends of the past price) and Technological Methods (use Data Mining Technologies, Artificial Neural Networks, Machine Learning etc.)

Raut Sushrut Deepak, Shinde Isha Uday, and Dr. D. Malathi from SRM University of India in their academic work Machine Learning Approach in Stock Market Prediction apply Machine Learning and ANN to predict stock values of Bombay Stock

[1] https://en.wikipedia.org/wiki/Stock_market

[2] https://en.wikipedia.org/wiki/Stock_market_prediction
~~~

Exchange[3]. They came to the conclusion that input data plays an important role in prediction along with machine learning techniques. Using SVM and RBF they reached accuracy up to 89%.

So, prediction of stock prices is a difficult task. There are people who believe they really can't be predicted, and it is just a guess. Some other people believe that human intuition is the most powerful tool for prediction. Others, again, believe that brokers accumulate knowledge and human intellect works with this accumulated data, figures out trends and gives a prediction without giving a detailed explanation.

In this project, I will try to create a model that works as a third example - finding trend within accumulated data.

I will use data from London Stock Exchange to predict stock prices for several companies. Data was obtained from Yahoo Finance.

For the characteristics of the dataset, I looked at the stock data for Marks and Spencer Group (MKS.L). The dataset in csv format (comma separated values) contains data for M&S stock from 4/1/2014 to 1/5/2018. There are 951 data points, and for each data point data includes the following: Date, Opening Price, High Price, Low Price, Closing Price, Adjusted Closing Price and Volume. I will predict Closing Price.

Problem Statement

Predicting a stock price is important because having a profit is efficient if we sell the stock at a higher price than we bought it. First, we need to buy a stock which will be rising. Second, in order to achieve maximum profit, we will not sell if the price will continue to go up. And the perfect time to sell is just before the price will go down.

So, the problem is to predict the future price the stock, having historical data for this stock. I will predict the stock price for the next trading day after the last date of my historical data.

One of the challenges of this project is that I will work with time series data, for this reason, train-test splitting can't be done with functions using shuffle. This would lead to the situation that algorithm would have to predict data from the middle of the dataset, actually being trained on data before and after the predicting data, which is absolutely incorrect. I need to train my algorithm only on the data before the predicting date, as in the real world I will have only these

[3] http://acadpubl.eu/jsi/2017-115-6-7/articles/8/12.pdf

data. I will need to split the dataset manually on the chronological basis. I will take prices for N days as features and price of the immediately next trading day as a label for these features, after this I will pass some data not using it for training.

I will apply linear regression algorithm to my data and predict the closing price for the next trading day. Having predictions for my test data I will compare them to actual closing prices for the same days and evaluate my algorithm using appropriate metric (see below). I will try to tune my algorithm using feature selection (e.g. vary the number of days before prediction).

Metrics

To quantify the performance we will use a root mean square error (RMSE), which is a frequently used metric to estimate the difference between predicted and observed values. We will use RMSE to evaluate the difference between the predicted stock price for the particular date and actual closing price for this date.

RMSE has some very important advantages for our project. The effect of an error will be proportional to the squared size of this error, so bigger error is more important for this metric, just as bigger errors in predictions are more important in making financial decisions. And small errors have a very small impact. Also squaring the error will ensure that errors for both overestimation and underestimation will be counted instead of neutralized.

II. Analysis

Data Exploration

The code for the data exploration, exploratory visualization, and visualization can be found in the notebook:

EDA.ipynb

A primary dataset used in this project is a dataset of stock prices for Marks and Spencer Group from London Stock Exchange (code MKS.L). The dataset was downloaded from Yahoo Finance, saved as a csv file and converted to Pandas dataframe. It contains 951 data points, from 4/1/2014 to 1/5/2018.
The sample of data:

	Date	Open	High	Low	Close	Adj Close	Volume
0	2014-04-01	452.000000	460.899994	452.000000	459.799988	385.056885	7221352.0
1	2014-04-02	460.100006	472.484985	460.100006	469.899994	393.515045	5419474.0
2	2014-04-03	469.899994	475.100006	469.899994	471.600006	394.938721	5786886.0

3	2014-04-04	463.600006	473.000000	460.799988	461.899994	386.815521	8489417.0
4	2014-04-07	458.600006	460.410004	359.200012	452.899994	379.278534	4168587.0

Every row is a data point, and columns contain following features:

Feature	Format of data	Description
Date	Datetime: YYYY-MM-DD	Trading date
Open	float 6 decimal places	Price of the stock when market opens on trading date
High	float 6 decimal places	The highest price of the stock during trading day
Low	float 6 decimal places	The lowest price of the stock during trading day
Close	float 6 decimal places	Price of the stock when market closes on trading date
Adj Close	float 6 decimal places	Adjusted closing price[4] - closing price of the stock on the trading date that has been amended to include any distributions and corporate actions that occurred at any time prior to the next day's open
Volume	float 1 decimal place	Number of shares traded on the trading date

Statistics for the dataset:

	Open	High	Low	Close	Adj Close	Volume
count	951.000000	951.000000	951.000000	951.000000	951.000000	951.0
mean	415.092534	419.303639	410.249037	414.812991	374.730233	6,687,027.0
std	78.815122	79.071253	78.615665	78.639503	59.082044	4,075,954.0
min	276.000000	299.000000	255.100006	285.200012	264.791504	400,006.0
25%	339.250000	341.959992	335.349991	338.600006	321.249939	4,135,072.0
50%	419.899994	423.200012	414.899994	418.500000	366.600311	5,805,118.0
75%	476.600006	481.650009	473.333008	476.949997	415.322662	7,907,527.0
max	595.000000	600.000000	592.500000	596.500000	520.689880	36,639,560.0

The count is the same for all the features, which means, that there are no missing values.

4 https://www.investopedia.com/terms/a/adjusted_closing_price.asp#ixzz53pEXD1WS

Min, max and mean values for the Open, Close, High and Low are very close, but for Adj Close values are significantly lower, which is expected due to the nature of the trading process and of the adjusting closing price definition.

Also, we have 1 NaN value in each column, it is data point 10/6/2017.

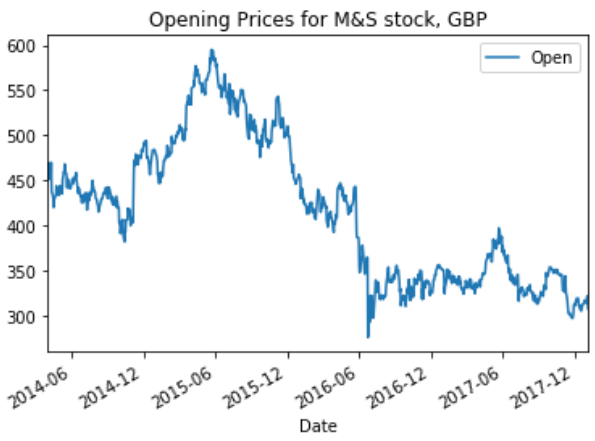
For better data exploration extra features were created:

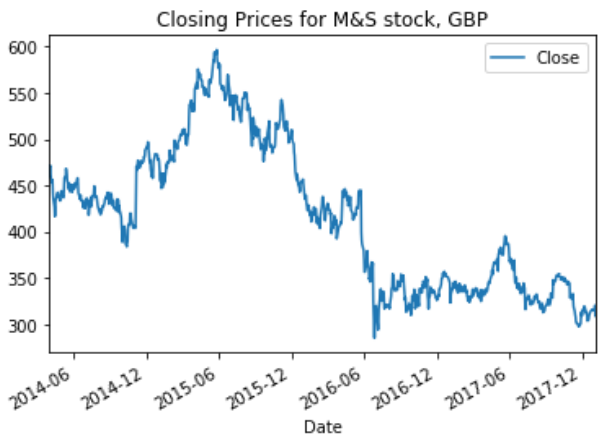
```
df.loc[:,'Daily Var'] = df.loc[:,'High'] - df.loc[:,'Low']
df.loc[:, 'Daily Change'] = df.loc[:,'Close'] - df.loc[:,'Open']
df.loc[:, 'Daily Change %'] = df.loc[:, 'Daily Change'] / df.loc[:, 'Open'] * 100
```

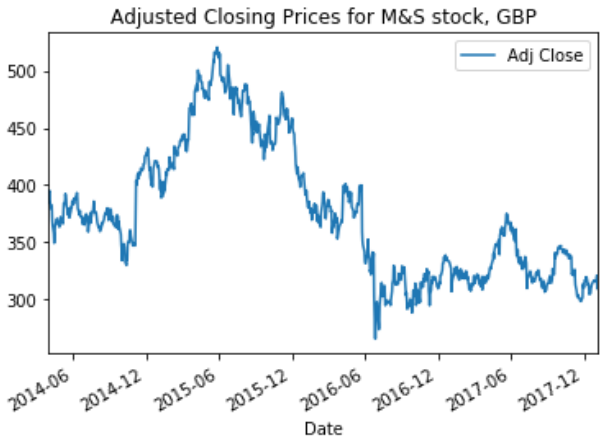
I believe features explorations can be done more easily with visualizations.

Exploratory Visualization

Let's explore Open, Close, Adj Close:

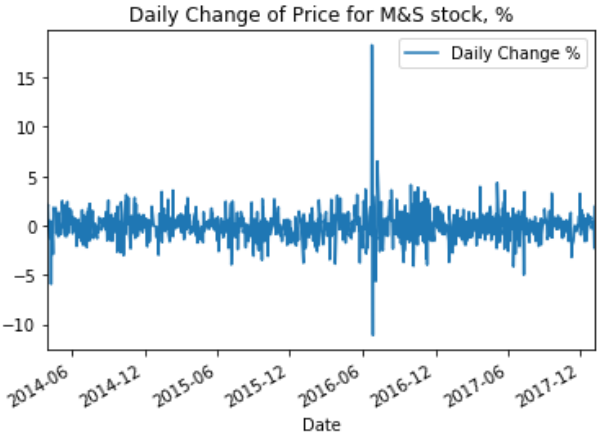


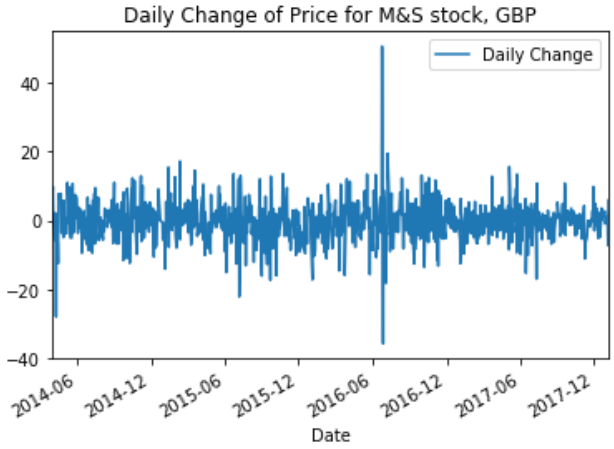




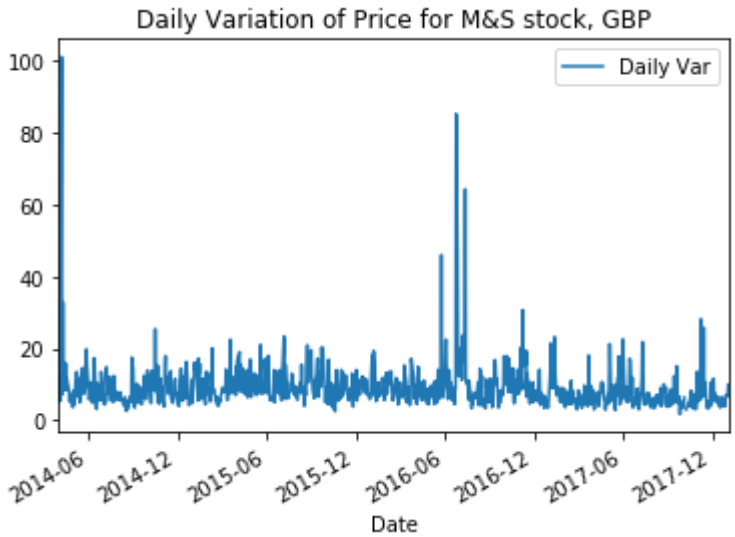
We can see very similar trends in all these prices.

It will be interesting to see also the daily change - the difference between closing and opening price:



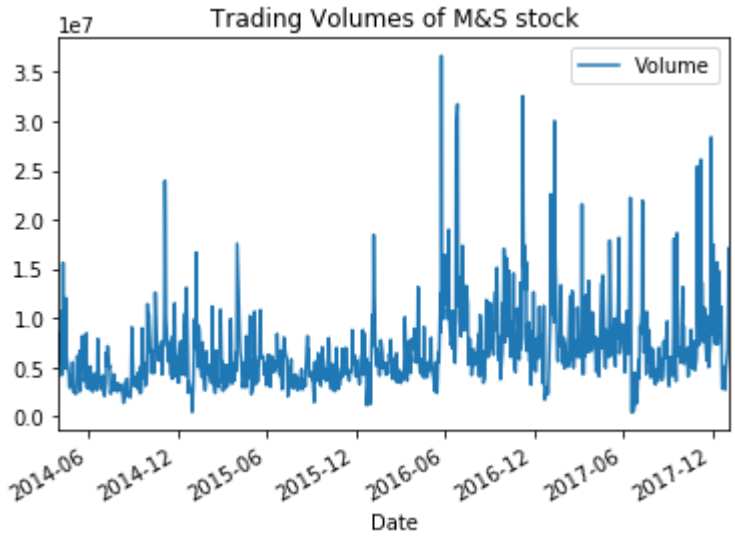


We can see that for the majority of dates price changed less than 20 GBP or less than 5%. Extremely big rise and fall of price within a day we can see the days when Brexit referendum was held, which definitely lead to an abnormal behavior on the stock exchange.
We can see also the daily variance, which will show us the difference between high and low price in the trading day:



We can see rather a similar trend. Most of the daily variations are up to 20 GBP, very high variations over 80 GBP happened on the days of Brexit referendum. But here I can also see an extremely high variation, over 100 GBP, on April the 7th, 2014. Price felt 100 GBP and then rise again. It is hard to find the reason, but maybe it can be connected to the call of the President of Czech Republic for NATO forces to enter Ukraine to prevent eastern expansion.

Finally, the Volume:



Trading volumes change a lot every day, but generally, we can see that they increased last 1,5 years.

Algorithms and Techniques

As mentioned before the signal-to-noise ratio in trading is low. Therefore, complicated models would overfit. A linear regression is appropriate for simplicity reasons.
The regression models I use for the predictions are the following:

- Linear Regression Regressor
  Explanation: we calculate the best-fitting line for the observed data by minimizing the sum of the squares of the vertical deviations from each data point to the line (if a point lies on the fitted line exactly, then its vertical deviation is 0). Having the deviations are first squared, before summing them, there are no cancellations between positive and negative values.[5]
- Linear Support Vector Machine Regressor
  Explanation: SVM is considered to be one of the most suitable algorithms available for time series prediction. The SVM involves plotting of data as a point in the space of n dimensions. These dimensions are attributes that are plotted on particular coordinates. In its simplest case, the binary classifier SVM algorithm draws a boundary over the data set called the hyperplane, which separates data into two. The hyperplane is the decision boundary which is laterextended or maximised on either side in between the data points.
- Ridge Regressor
  Explanation: Ridge regression addresses some of the problems of Ordinary Least Squares by imposing a penalty on the size of coefficients. The ridge

[5] http://www.stat.yale.edu/Courses/1997-98/101/linreg.htm

coefficients minimize a penalized residual sum of squares:

$$\min_{w} ||Xw - y||_2^2 + \alpha||w||_2^2$$

Here, α>= 0 is a complexity parameter that controls the amount of shrinkage: the larger the value of α, the greater the amount of shrinkage and thus the coefficients become more robust to collinearity.[6]

For tuning the Linear Regression model, we will use the **Grid Search** technique.
To evaluate our models, we will use as our metrics the **Root Mean Square Error**.
We will also look at the

- Root Mean Squared Percentage Error
- Mean Absolute Error
- Explained Variance Score
- Mean Squared Error
- R2 score

For splitting training/test set and because of the nature of the data (time series data) we cannot use the out of the box sklearn's train_test_split function which shuffles the data with consequence to lose the influence of the older values to the recent ones. Also, If the data were shuffled, e.g. the close price for 1 Sept 2016 might be in the training set. We might then be asked to predict the close price for the next day after 31 Aug 2016, that will be the price for 1 Sept 2016 which we'd have seen before.
Therefore, we would need to develop a custom algorithm for our dataset to split it into train and test set with respect to the chronological order of our data.
In ratio to the KFold cross-validation technique and with respect to the sequential nature of data, we also need to develop a custom algorithm to use it for cross-validation.

Benchmark

I will use custom persistence algorithm as a benchmark model. I will test it on the same data as my primary model, and I will compare the results. Ideally, my final model will outperform the persistence model.

The persistence algorithm will be returning the latest closing price for each data point: Consider that our data points are vectors of 10 sequential closing prices (i+0, i+1, …, i+9), the return of our persistence function will be the i+9 closing price.

The persistence algorithm is naive. It is often called the naive forecast. It assumes the least about the specifics of the time series problem to which it is applied. So, because of its simplicity, I choose it as my benchmark model.

[6] http://scikit-learn.org/stable/modules/linear_model.html#ridge-regression

III. Methodology

Data Preprocessing

During the data exploration, we found a very small number of NaN values. We decided to remove those records from the dataset. Also, we observed increased volatility around significant political events like the Brexit referendum. We decided to keep those trading periods in the dataset because they reflect the strong and permanent relation between stock market and politics.

Data Exploration has revealed the relationship between all the features of the data set with the Close price. While we made a few experiments with transformed and engineered features we finally decided to use only the Close price because it seemed that for the rest of the features their contribution to our model's performance was poor to pay back for the complexity they added.

The technical analysis features I engineered during the EDA were the ones below:

- Average Stock’s Range (daily High minus Low)
- Average Stock’s Daily Change (Close minus Open)
- Average Stock’s Daily Change percentage (Close minus Open, divided by Open)

As the EDA implied and as we see in the grid search showed that the influence of the technical analysis features is not worthwhile.

So, the refined set of features is a set of 15 sequential closing prices.

Implementation

The code for the data preprocessing, processing and training, and visualization can be found in the notebook:

Features-and-model-MKS.ipynb

I initially implemented the Linear Regression algorithm with the following basic features:

Close prices on each of the n days prior to the first prediction date

Process:

- Construct dataframe X containing initial features and dataframe y with Closing prices. This required some boilerplate code to extract the relevant features from the dataset and put them in an appropriately formatted dataframe
- Split X and y into training and test datasets
  Wrote my own function to do this (split_train_test_set) instead of using sklearn's train_test_split. This was because sklearn's function automatically shuffles the data. Shuffling the data is not desired for situations in which data is ordered

In order to respect the sequentially of the data points, I needed to split the dataset into two sequential sub-sets, return the first one for training and the second for testing.
So, the point was to find the specific index that separates the set into 2 sub sets: so, supposed that X is the data set and the test size (in [0,1]) is the size of the test set then:

```
split_index = int(len(X) * (1-test_size))
X_train = X[:split_index]
X_test = X[split_index:]
y_train = y[:split_index]
y_test = y[split_index:]
```

- Train model on training data.
- Also developed a create_cv_sets method to cut the dataset into cross-validation folds BUT with respect to their chorological order
  So, the method takes as argument the features and the target sets (X, y), the number of splits and the size of the Fold. The challenge is to find the split index which is splits the sets like that

```
def create_cv_sets(X, y, n_splits, size):
    cv_sets = []
    for index in range(n_splits):
        X_train, X_test, y_train, y_test = split_train_test_set(X[index:index + size], y[index:index + size])
        cv_sets.append([X_train, X_test, y_train, y_test])
    return cv_setsPredict prices on test features
```

- Print metrics
- Model, train and prediction
- For all the three model I used the sklearn python library and specifically for:
  - Linear Regression
    - Imported the LinearRegression from sklearn.linear_model
    - Initialized the LinearRegression with the default parameters (fit_intercept=True, normalize=False, copy_X=True, n_jobs=1)
    - Imported the GridSearchCV from sklearn.model_selection
    - Imported the grid_searchfrom sklearn
    - Used the GridSearchCV for tuning the model against the parameters “fit_intercept”, “normalize”, “copy_X”
      See below the values table with the gridsearched parameters

fit_intercept	normalize	copy_X
TRUE	TRUE	TRUE
TRUE	TRUE	FALSE
TRUE	FALSE	TRUE
TRUE	FALSE	FALSE
FALSE	TRUE	TRUE
FALSE	TRUE	FALSE
FALSE	FALSE	TRUE
FALSE	FALSE	FALSE

    - Fit the model with the train set
    - Predict with the test set
  - Ridge Regression

        - Imported Ridge from sklearn.linear_model
        - Initialized Ridge with the default parameters (alpha=1.0, copy_X=True, fit_intercept=True, max_iter=None, normalize=False, random_state=None, solver='auto', tol=0.001)
        - Fit the model with the train set
        - Predict with the test set
    - Linear Support Vector Machine Regression
        - imported LinearSVR from sklearn.svm
        - initialized LinearSVR with default parameters (C=1.0, dual=True, epsilon=0.0, fit_intercept=True, intercept_scaling=1.0, loss='epsilon_insensitive', max_iter=1000, random_state=0, tol=0.0001, verbose=0)

- error visualizations, actual vs prediction visual comparison plots
    - imported and used pyplot from the matplotlib lib

The coding of the solution was straightforward. Using the python's sklearn lib I found no complications. The only thing I missed was the grid search tuning algorithm for my custom parameters. That algorithm, I had to develop with my own devices, I needed to nest 3 for loops, one for each parameter, iterating their possible values. (The code can be found in the notebook “features-and-model-MKS-grid-search.ipynb”)

Refinement

As an improvement of the model, I explored the options of adding features to the training/test dataset like daily change (the difference between close/open value) and the trading volume. The influence of these new features was not meaningful, something that confirmed the intuition we had from the EDA.
I also explored the option to enlarge the dataset, so I acquired the full history of prices that we could find in Yahoo Finance (since 1990-01-01 until today) and by using the same process we resulted in much better performance for our models. The wider dataset gave us the opportunity to have more cross-validation folds.

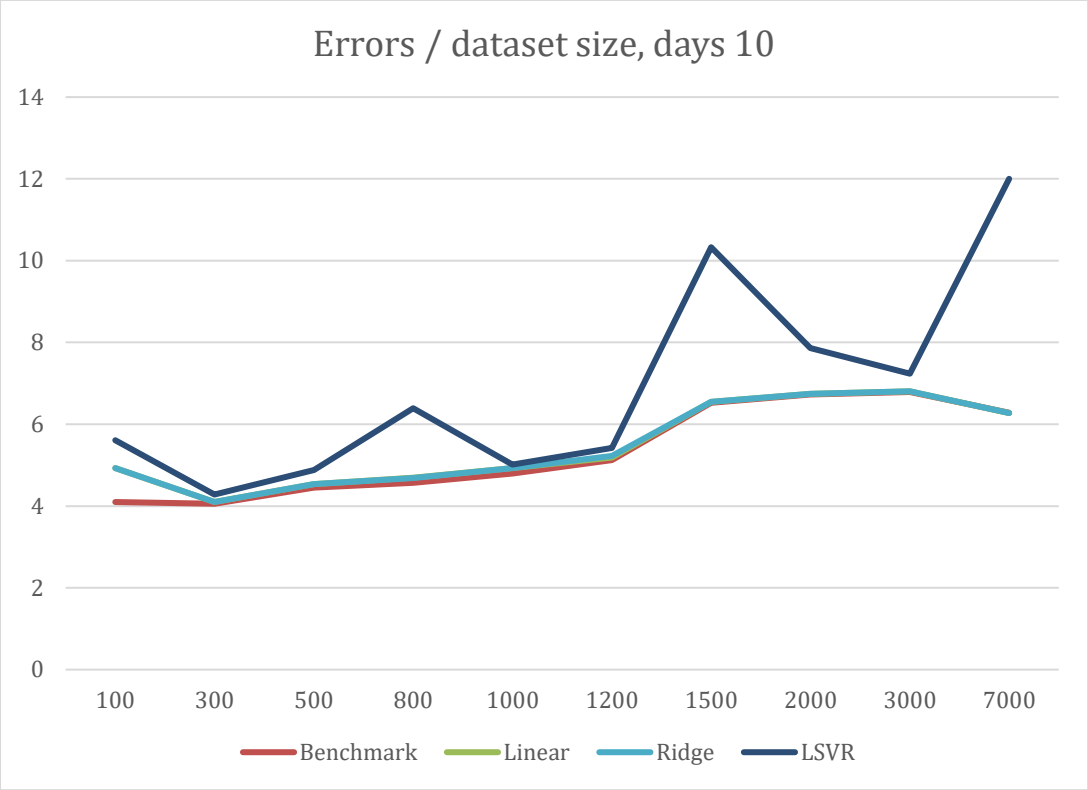
I handcrafted grid search method to tune the models against my custom parameters (usage of technical analysis features, number of days, size of the dataset). We can find the code in the notebook “features-and-model-MKS-grid-search.ipynb”.
Below we can see the result of the method. The optimal Linear Regressor is highlighted. The parameters no technical analysis features, 10 days, 300 data points demonstrate the less RMSE for the Linear Regressor.

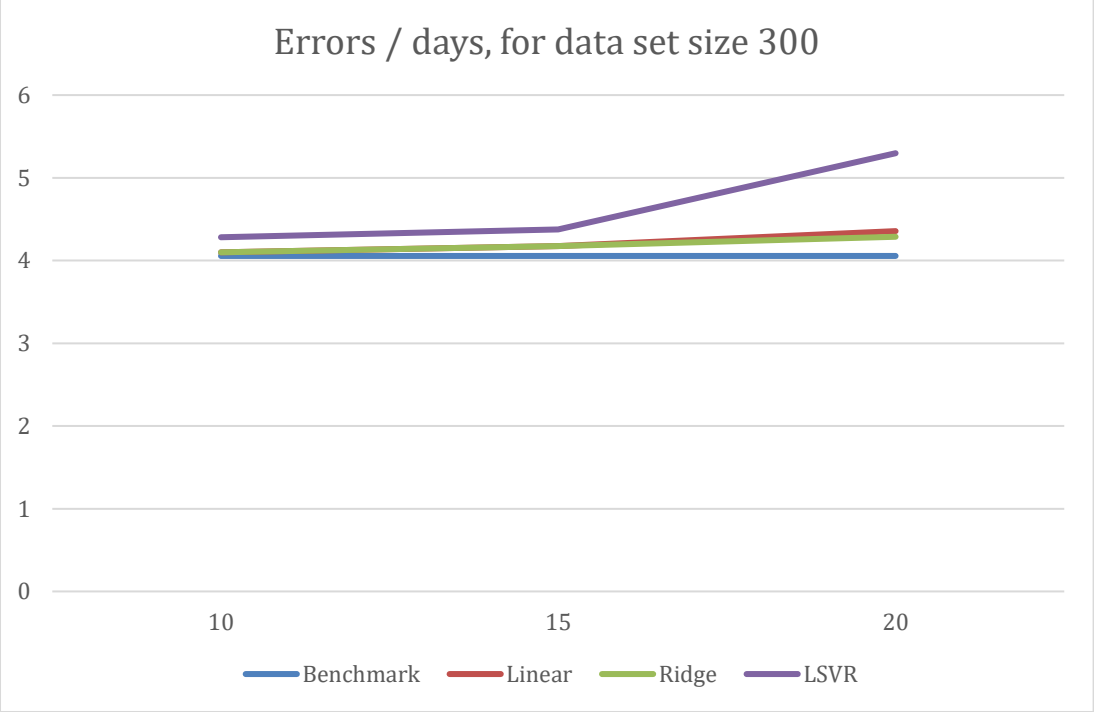
UseTA	days	X_size	Benchmark	Linear	Linear_Train	Ridge	Ridge_Train	LSVR	LSVR_Train

TRUE	10	100	4.09834	5.04258	3.60864	5.06587	3.44433	4.95926	4.7881
TRUE	10	300	4.05541	4.13512	4.98057	4.10876	4.9684	4.65662	5.50229
TRUE	10	500	4.44984	4.77824	7.01418	4.78602	7.01431	6.4426	8.46679
TRUE	10	800	4.56521	4.83362	7.37212	4.80212	7.37316	4.73106	7.42157
TRUE	10	1000	4.79599	5.08474	7.213	5.04733	7.21312	11.8296	16.2549
TRUE	10	1200	5.12383	5.32725	7.26221	5.357	7.25792	5.72222	7.96244
TRUE	10	1500	6.52561	6.54513	6.38926	6.54922	6.38931	7.96373	8.38607
TRUE	10	2000	6.72916	6.72558	6.04491	6.72431	6.04089	6.91142	6.45929
TRUE	10	3000	6.78857	6.82093	7.75065	6.82091	7.75065	7.15156	8.0698
TRUE	10	7000	6.28203	6.28725	8.02582	6.28425	8.02188	6.29898	8.03557
TRUE	15	100	4.09834	5.24153	3.51486	6.13333	3.25942	8.80136	9.85516
TRUE	15	300	4.05541	4.40692	4.92092	4.19052	4.90725	9.09748	9.91199
TRUE	15	500	4.44984	4.71341	7.0239	4.72163	7.0241	5.43707	8.01474
TRUE	15	800	4.56521	4.82629	7.33146	4.78621	7.3328	4.71203	7.37611
TRUE	15	1000	4.79599	5.06197	7.17593	5.01515	7.17693	6.3115	8.95265
TRUE	15	1200	5.12383	5.23092	7.23718	5.35309	7.22912	7.66408	10.7086
TRUE	15	1500	6.52561	6.49985	6.3718	6.55606	6.35774	6.60472	6.61249
TRUE	15	2000	6.72916	6.75246	6.02895	6.75325	6.02437	7.09895	6.32495
TRUE	15	3000	6.78857	6.82185	7.73727	6.82185	7.73727	11.7126	12.7732
TRUE	15	7000	6.28203	6.29693	8.02005	6.29412	8.0165	10.58	12.6121
TRUE	20	100	4.09834	4.84827	3.36852	6.63131	2.98973	9.1137	9.94652
TRUE	20	300	4.05541	4.39414	4.80965	4.28455	4.82767	10.2193	10.7462
TRUE	20	500	4.44984	4.67984	6.94482	4.70707	6.94935	4.88654	7.22405
TRUE	20	800	4.56521	4.8749	7.28383	4.81388	7.28554	7.9018	11.1327
TRUE	20	1000	4.79599	5.0701	7.13477	4.99914	7.13758	5.58974	8.0507
TRUE	20	1200	5.12383	5.29201	7.19088	5.41656	7.1835	5.17652	7.23071
TRUE	20	1500	6.52561	6.57799	6.32337	6.5768	6.32337	6.67029	6.73549
TRUE	20	2000	6.72916	6.76287	6.01575	6.75585	6.01133	6.90669	6.12649
TRUE	20	3000	6.78857	6.82646	7.73448	6.82647	7.73448	8.14192	8.97802
TRUE	20	7000	6.28203	6.3092	8.00785	6.30667	8.00465	6.37694	8.0905
FALSE	10	100	4.09834	4.93059	3.53602	4.92935	3.53602	5.61206	4.471
FALSE	10	300	4.05541	4.09929	4.97364	4.09933	4.97364	4.28128	5.13229
FALSE	10	500	4.44984	4.53445	7.12103	4.53443	7.12103	4.88405	7.55269
FALSE	10	800	4.56521	4.69	7.42478	4.68737	7.42351	6.39394	9.44733
FALSE	10	1000	4.79599	4.931	7.26284	4.93113	7.26045	5.01175	7.33761
FALSE	10	1200	5.12383	5.18457	7.30051	5.22544	7.29355	5.41926	7.57595
FALSE	10	1500	6.52561	6.54894	6.39181	6.54895	6.39181	10.331	11.8366
FALSE	10	2000	6.72916	6.74474	6.05416	6.74106	6.04998	7.86057	7.86411
FALSE	10	3000	6.78857	6.80571	7.76451	6.80571	7.76451	7.2366	8.22254
FALSE	10	7000	6.28203	6.27628	8.03093	6.27628	8.03093	12.0021	14.1893
FALSE	15	100	4.09834	5.0657	3.45755	5.06533	3.45756	8.04222	6.42001
FALSE	15	300	4.05541	4.17453	4.91491	4.17457	4.91491	4.37636	5.04596
FALSE	15	500	4.44984	4.54616	7.09843	4.54615	7.09843	4.54741	7.14215
FALSE	15	800	4.56521	4.70329	7.37708	4.70045	7.37592	5.02536	7.7556
FALSE	15	1000	4.79599	4.92462	7.22675	4.92525	7.22411	5.4849	7.87915
FALSE	15	1200	5.12383	5.17862	7.26739	5.23583	7.25863	5.19583	7.27495
FALSE	15	1500	6.52561	6.51709	6.38148	6.55082	6.36861	7.7553	8.52298
FALSE	15	2000	6.72916	6.76299	6.03755	6.7604	6.03237	6.87229	6.36271
FALSE	15	3000	6.78857	6.79881	7.75291	6.79881	7.75291	7.69653	8.66708
FALSE	15	7000	6.28203	6.2859	8.02566	6.2859	8.02566	6.31034	8.03942
FALSE	20	100	4.09834	4.78997	3.29471	4.78963	3.29472	8.68358	7.01571
FALSE	20	300	4.05541	4.35568	4.87936	4.28721	4.83278	5.29717	5.98946
FALSE	20	500	4.44984	4.57592	7.03185	4.57589	7.03185	4.79939	7.37089
FALSE	20	800	4.56521	4.73294	7.32884	4.72953	7.32735	4.71871	7.3461
FALSE	20	1000	4.79599	4.92021	7.1841	4.91953	7.18184	6.13787	8.70337

FALSE	20	1200	5.12383	5.18069	7.2279	5.22457	7.22093	5.68007	7.8944
FALSE	20	1500	6.52561	6.5363	6.34439	6.56891	6.33269	6.62142	6.57026
FALSE	20	2000	6.72916	6.76777	6.02338	6.76342	6.0193	13.2286	14.0768
FALSE	20	3000	6.78857	6.80264	7.7499	6.80264	7.7499	7.062	8.01024
FALSE	20	7000	6.28203	6.30255	8.0147	6.30255	8.0147	10.5649	12.6689

In the graphs below, we can see visualization of the grid search results





In the graphs below, we can see the benefit of the richer data set. We managed to close the gap between the benchmark model and our Linear Regressor. In addition, the wider KFold cross validation gave us more confidence about the performance of our model.

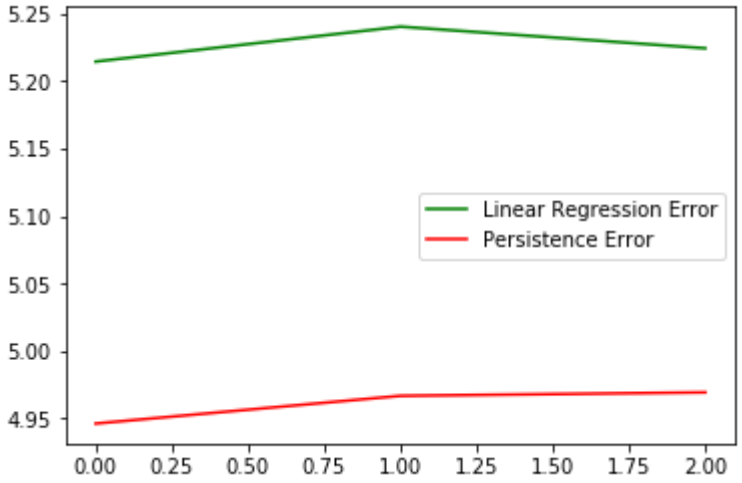


Figure 1 Errors in the Initial Dataset for each fold

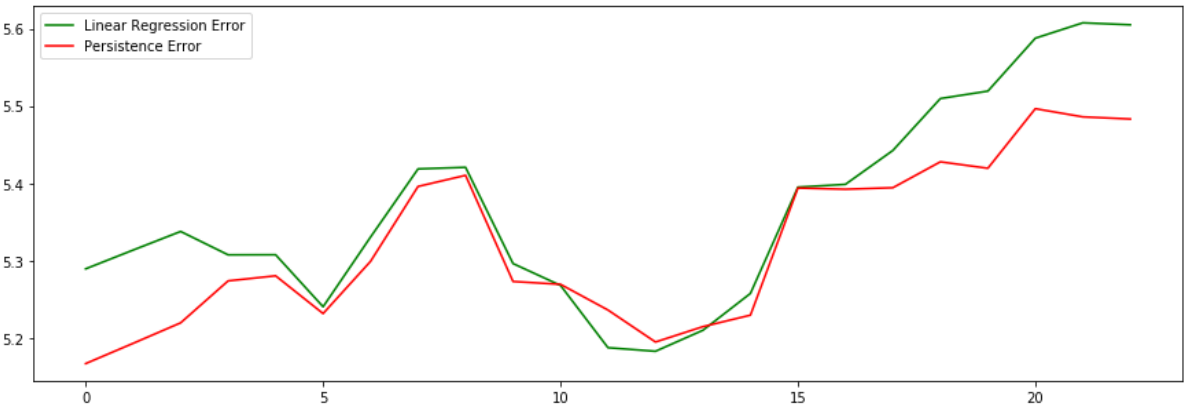


Figure 2 Errors in the Full Dataset for each fold

The last part of the processing code with the custom KFold cross validation and the visualization can be found in the notebook “features-and-model-MKS-final.ipynb”

IV. Results

Model Evaluation and Validation

So, our final model should be the linear regression model, trained with the full dataset, using as features 10 sequential closing prices. The solution specific parameters such as the use of technical analysis features or not, the number of features and the number of data points were grid searched with a custom method. The model is tested against the 30% of the dataset as it was split with 70/30 ratio to train/test.

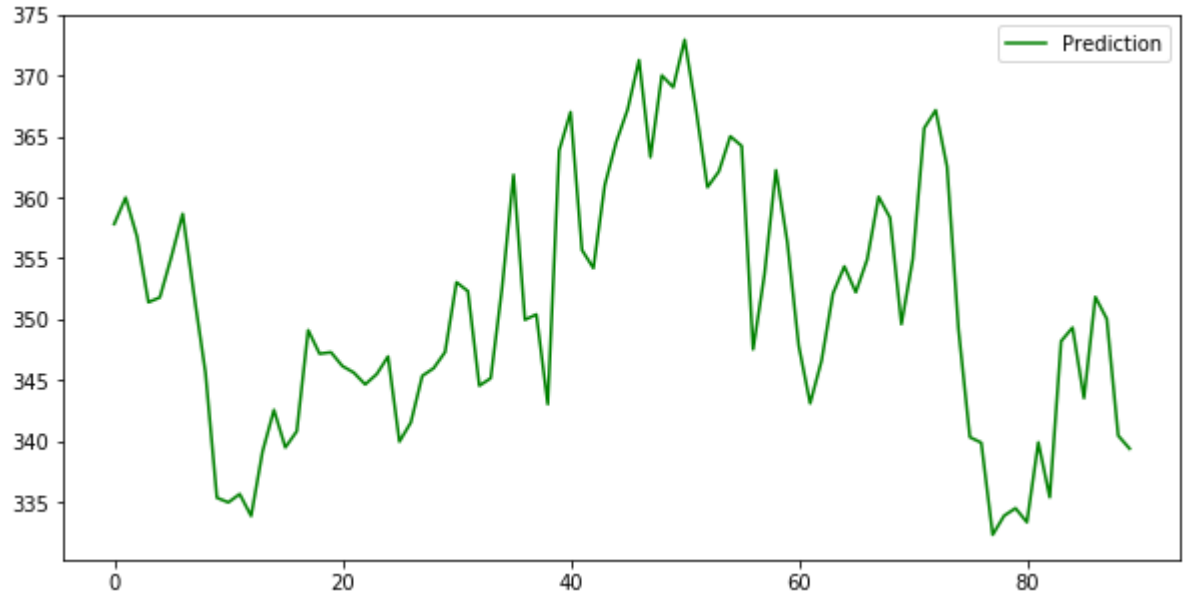


Figure 3 Predicting Future Closing prices using the Regression Model

Justification

Overall, this model on average does not perform better than the benchmark model of predicting with at max 5.6 rmse the stock's closing price for the next day.

The solution gives reasonably accurate predictions but it is not significant enough to reliably give advice on trades because a 5.6% error is significant in trading. There are also transaction costs with every trade, which would cut into profits.

Comparing our model's performance with the benchmark model gives very similar performance, in many folds performs better. We can see the comparison graph below:

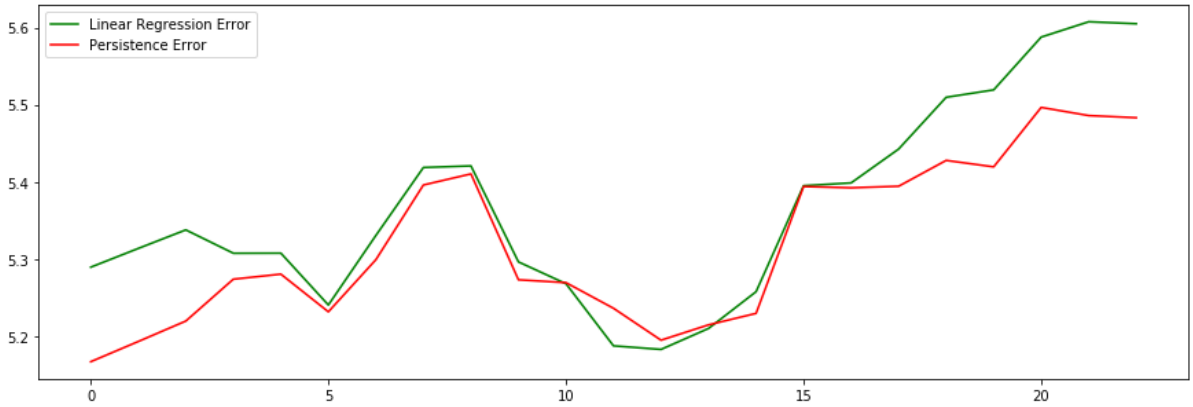


Figure 4 RMSE against Folds

Obviously, the comparison is not so eager to our Linear Regression Model

V. Conclusion

Free-Form Visualization

The graph below visualizes the predictions compared with the actual close prices. The purpose is to see how predictions vary with actual prices.

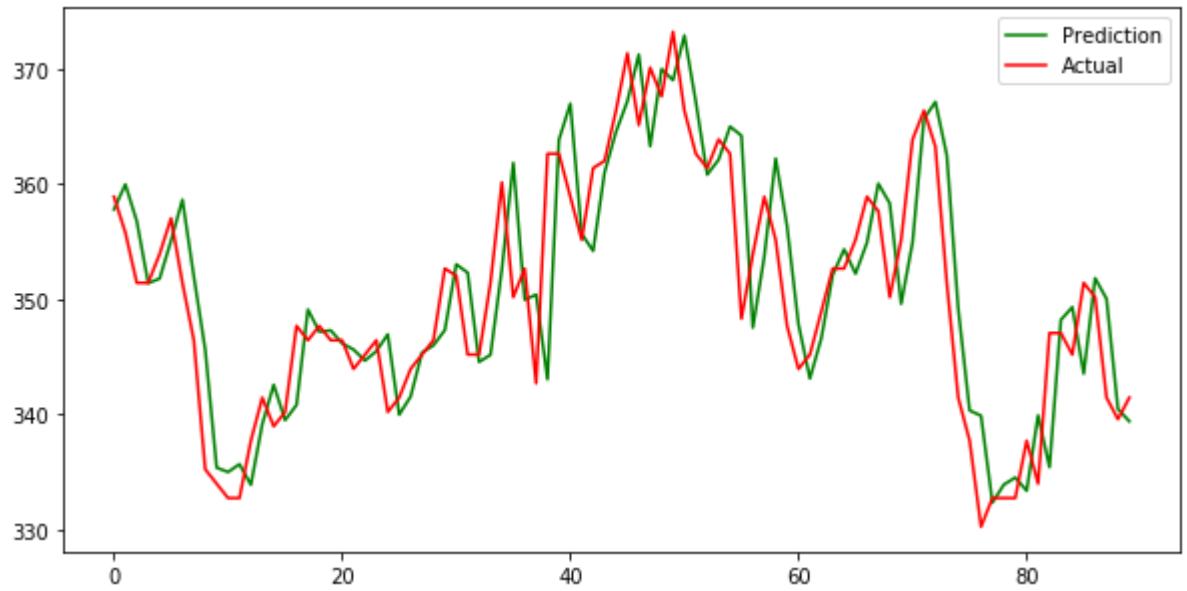


Figure 5 Actual Closing Price vs Predicted Closing Price

The graph below visualizes the test errors against the several cross-validation folds

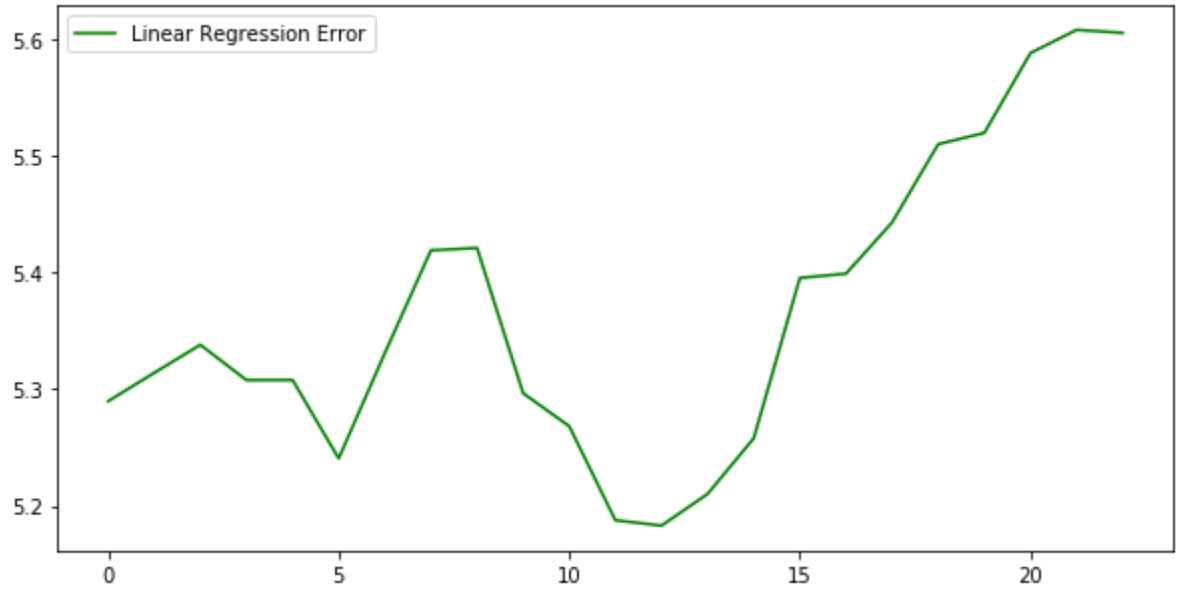


Figure 6 RMSE against the cross-validation folds

Reflection

The problem of stock market prediction is one which continues to allure researchers every day. In my project, I collected data from Yahoo Finance using

the pandas datareader, after EDA processes I decided to use Linear Regression to model the stock market prediction. In the data pre-processing phase I engineered model features and, finally, I implemented 3 models (Linear Regression, Ridge Regression, Linear Support Vector Machine) and a Persistence model (used as the benchmark model). As a UI, I created a jupyter notebook that loads the model, downloads the latest 10 stock prices and predicts the future price.

Given the powerful toolbox of a sklearn python library, the implementation of the ML regression models was straightforward. Although we found challenging the data preprocessing phase. We needed to form the pandas dataframes with the selected features. We also needed to implement the train/test set split logic because we didn't want to use sklearn's TimeSeriesSplit method.

The GridSearchCV method was a very handful in tuning the model parameters but we also had to manual grid search the usage or not of technical analysis features, the number of features and the data points used for the model training.

The prediction models developed in this project were based on the linear regression and demonstrate remarkable accuracy. However, and given that the benchmark model gives very similar RMSE I would suggest for further improvements.

Improvement

We made several experiments while we were investigating ways to improve the performance of the models: We included more features (like daily variation, volume, more historical days), we increased the number of training/testing data points without significant improvement.
However, we can still try some improvements like the ones below:

- adding features from the FTSE index and the group of retail stocks index
- do more data pre-processing trying to eliminate the Brexit effect by removing the data points of the period with the very high market uncertainty. (Of course, this would reduce the scope of the project)

Other improvements but maybe out of the scope of the project can be the ones below:

- we can employ LSTM NN model which is also suitable for predicting time series data
- introduce features based on the trader's intuition
- use sentiment analysis for the news related to the stock in order to produce features