Predicting Daily Adjusted Close Stock Prices (SPY) using Time-series and XGBoost methods

Bao Nguyen

DEFINITION

Project Overview

Predicting stock prices has been a long-term interest in finance. For example, searching the phrase 'predict stock prices' on Google Scholar returns 1,020,000 results. There have been many machine learning and artificial intelligence methods used to predict stock prices. Wang, YF (2002) develops a fuzzy grey prediction to predict a large volume of stock prices in real-time. Gidófalvi, G (2001) uses a naïve Bayesian text classifier and leverages news articles to predict stock price movements. Roondiwala, M, Patel, H, and Varma, S (2015) predict stock prices using Long short-term memory (LSTM). Although predicting stock prices to match the market movements exactly might be impossible, attempts to predict stock prices are still valuable to understand and acknowledge the challenges.

Studying Economics motivates me to do this project. I have been using time-series and econometrics methods during my study and research in economics-related topics; however, applying those methods into predicting stock prices carry a lot of real-world applications.

I used the yahoofinancials package to gather SPY stock prices from Yahoo Finance. The inputs are adjusted close prices of SPY with the period from 2019-01-01 to 2019-12-31.

Problem Statement

I will build a stock price predictor that gives an estimation of 7-day stock prices for the S&P 500 Index (SPY). The dependent variable is the adjusted close prices.

I will use XGBoost (stands for eXtreme Gradient Boosting) to predict stock prices. According to Basak et al. (2018), given the nature of boosting (combining the results of various weak predictors to make a strong prediction), XGBoost approximates regressors to the training samples and finds the best split of the aggregate of the regressor functions.

Evaluation Metrics

The evaluation metric is RMSE (Root mean squared error), which measures the errors between the stock's actual prices and the predicted adjusted close prices. Taking the square of the errors avoids the errors from underestimation and overestimation cancel out each other. The smaller the RMSE is, the better the prediction would be.

ANALYSIS

Data Exploration

The dataset has 251 rows. The earliest date is 2019-01-02. The latest date is 2019-30-12. Here is the explanation of the variables. 'Adjclose' (adjusted close price) is the main variable of interest a.k.a. the dependent variable.

Column	Format	Meaning
Date	YYYY-MM-DD	
high	Float	Maximum price of the stock during the trading day in USD.
Low	Float	Minimum price of the stock during the trading day in USD.
Open	Float	Price of stock when the market opened on that day in USD.
Close	Float	Price of stock when the market closed on that day in USD.
Volume	Integer	The number of shares of that stock traded in that day.
Adjclose	Float	The closing price of the stock after adjustments for all applicable splits and dividend contributions (in USD).

This is the snapshot of my dataset.

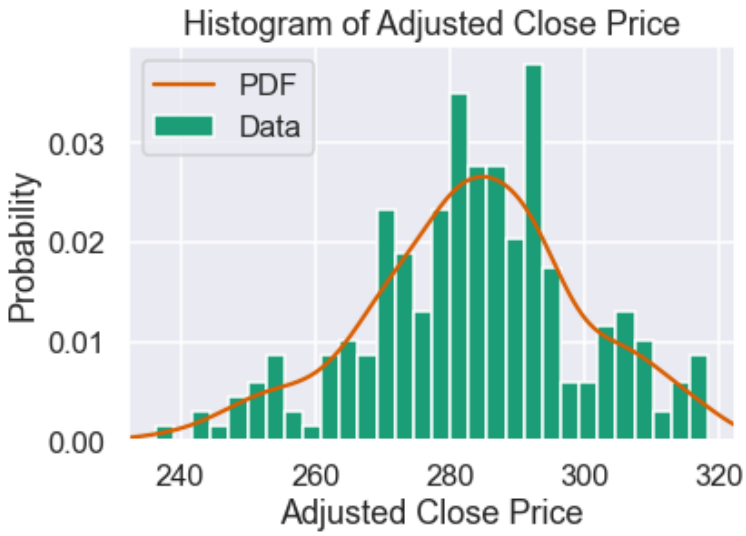
	date	high	low	open	close	volume	adjclose
0	2019-01-02	251.210007	245.949997	245.979996	250.179993	126925200	242.056915
1	2019-01-03	248.570007	243.669998	248.229996	244.210007	144140700	236.280746
2	2019-01-04	253.110001	247.169998	247.589996	252.389999	142628800	244.195160
3	2019-01-07	255.949997	251.690002	252.690002	254.380005	103139100	246.120560
4	2019-01-08	257.309998	254.000000	256.820007	256.769989	102512600	248.432953

The picture below shows the summary statistics of the dataset. The mean value for adjusted close price is USD $283.62. The minimum value of adjusted close price is USD $236.28. The maximum value of adjusted close price is USD $318.37.

	high	low	open	close	volume	adjclose
count	251.000000	251.000000	251.000000	251.000000	2.510000e+02	251.000000
mean	291.933625	289.426374	290.735538	290.872629	7.008375e+07	283.615941
std	14.891401	15.318617	15.201130	15.009474	2.600331e+07	15.948296
min	248.570007	243.669998	245.979996	244.210007	2.027000e+07	236.280746
25%	283.099991	280.324997	281.794998	281.444992	5.087125e+07	273.491623
50%	292.779999	289.950012	291.309998	291.500000	6.503020e+07	283.987000
75%	300.700012	298.369995	299.744995	299.830002	8.328380e+07	293.155945
max	323.799988	322.279999	323.739990	322.940002	1.787454e+08	318.371063

Exploratory Visualization

To further understand the distribution of adjusted close price, I draw a histogram (picture below). In general, the distribution of adjusted close price follows a normal distribution despite a few outliers.



This is adjusted close price through time. It was around USD $250 at the beginning of the year and reached around $320 in the end of the year.

Algorithms and Techniques

XGBoost is an ensemble learning method. Sometimes, it is not sufficient to use only one machine learning model. Ensemble learning offers a comprehensive solution by combining the predictive power of multiple learners. Bagging and boosting are two commonly used ensemble learners. XGBoost is a boosting algorithm. According to Daumé III (2019), boosting is a process of taking a weak learner and turning it into a strong learner (a strong learning algorithm). There are three

main features of XGBoost that make the algorithm interesting, namely, regularization penalize complex models to help prevent overfitting), handling sparse data (incorporate a sparsity-aware split finding algorithm to handle various types of sparsity patterns in the data), and weighted quantile sketch (use a distributed weighted quantile sketch algorithm to effectively handle weighted data) (Bhaskar Sundaram, 2020).

Since we are using XGBoost for time-series forecasting, there are a few terminologies that are specifically relevant to our problem. First, we have to define Horizon, which is the number of days in which the stock prices needed to be predicted. Second, since stock price today depends significantly on the previous days' stock prices, we have to pay attention to the number of lag features. Third, there are time-series related features. For instance, the stock price might depend on the day or month of the year, such as the month that the budget is released. Finally, stock prices might need to be normalized using mean and standard deviation values before sending them as inputs to the model. All of these considerations would be implemented later in the paper.

Benchmark

I use a simple moving average as a benchmark model. A moving average is a technique to smooth out time-series data to reduce the "noise", taking the average of a certain number of previous periods to come up with a "moving average" for a given period. I used 7 lags for my benchmark model. The model uses the previous 7-day values to predict the value of the following day. This is the graph representing the actual values and the predicted values.

The RMSE value of the 7-day SMA is 2.76. While this is considered not a good value, this provides some benchmarking value so that we can compare the RMSE results of other models with this benchmarking RMSE.

METHODOLOGY

Data Preprocessing

First, I will do feature engineering by creating time-series features: day of week, week, quarter, month, year, day of year, day, and week of year. This is how my dataset looks like after adding those features.

	date	high	low	open	close	volume	adjclose	dayofweek	quarter	month	year	dayofyear	dayofmonth	weekofyear
0	2019-01-02	251.210007	245.949997	245.979996	250.179993	126925200	242.056915	2	1	1	2019	2	2	1
1	2019-01-03	248.570007	243.669998	248.229996	244.210007	144140700	236.280746	3	1	1	2019	3	3	1
2	2019-01-04	253.110001	247.169998	247.589996	252.389999	142628800	244.195160	4	1	1	2019	4	4	1
3	2019-01-07	255.949997	251.690002	252.690002	254.380005	103139100	246.120560	0	1	1	2019	7	7	2
4	2019-01-08	257.309998	254.000000	256.820007	256.769989	102512600	248.432953	1	1	1	2019	8	8	2

Next, I will add the lags to the dataset. The add_lags function is inspired by the article 'Stock Price Prediction: XGBoost' on Medium. I use the add_lags function to add the lags. I choose a quite high value (15 lags). The nth lag column contains the stock price n days before the current timestamp. For instance, the first lag column contains yesterday's stock price value in today's row. Here is the dataset after adding the lags.

	date	high	low	open	close	volume	adjclose	dayofweek	quarter	month	...	adjclose_lag_7	adjclose_lag_8	adjclose_lag
0	2019-01-02	251.210007	245.949997	245.979996	250.179993	126925200	242.056915	2	1	1	...	NaN	NaN	Na
1	2019-01-03	248.570007	243.669998	248.229996	244.210007	144140700	236.280746	3	1	1	...	NaN	NaN	Na
2	2019-01-04	253.110001	247.169998	247.589996	252.389999	142628800	244.195160	4	1	1	...	NaN	NaN	Na
3	2019-01-07	255.949997	251.690002	252.690002	254.380005	103139100	246.120560	0	1	1	...	NaN	NaN	Na
4	2019-01-08	257.309998	254.000000	256.820007	256.769989	102512600	248.432953	1	1	1	...	NaN	NaN	Na

However, not all features are necessary for the model. Some features might provide little to no additional value. Therefore, I plot the correlation matrix to determine which features should be included in the model. Here is the result:

```
adjclose           1.000000
close              0.998875
high               0.997094
low                0.996090
open               0.993751
adjclose_lag_1     0.990438
adjclose_lag_2     0.982415
adjclose_lag_3     0.974873
adjclose_lag_4     0.966999
adjclose_lag_5     0.958958
adjclose_lag_6     0.951949
adjclose_lag_7     0.946730
adjclose_lag_8     0.938053
adjclose_lag_9     0.929108
adjclose_lag_10    0.918137
dayofyear          0.917267
month              0.911748
adjclose_lag_11    0.908077
adjclose_lag_12    0.899958
adjclose_lag_13    0.890797
week               0.888675
weekofyear         0.888675
adjclose_lag_14    0.880999
quarter            0.876276
adjclose_lag_15    0.870037
dayofmonth         0.102614
dayofweek         -0.022683
volume            -0.456453
year                    NaN
Name: adjclose, dtype: float64
```

I will choose only features whose correlation is greater than 0.90. Hence, the final features to include in the model are: adjclose_lag_1 to adjclose_lag_11, day of year, and month. I drop all other features. Here is the dataset after choosing important features.

	date	high	low	open	close	adjclose	month	dayofyear	adjclose_lag_1	adjclose_lag_2	adjclose_lag_3	adjclose_lag_4	adjclo
0	2019-01-02	251.210007	245.949997	245.979996	250.179993	242.056915	1	2	NaN	NaN	NaN	NaN	
1	2019-01-03	248.570007	243.669998	248.229996	244.210007	236.280746	1	3	242.056915	NaN	NaN	NaN	
2	2019-01-04	253.110001	247.169998	247.589996	252.389999	244.195160	1	4	236.280746	242.056915	NaN	NaN	
3	2019-01-07	255.949997	251.690002	252.690002	254.380005	246.120560	1	7	244.195160	236.280746	242.056915	NaN	
4	2019-01-08	257.309998	254.000000	256.820007	256.769989	248.432953	1	8	246.120560	244.195160	236.280746	242.056915	

Since there are inevitably many NaN values after implementing the lags, I will drop all rows with NaN values:

	date	high	low	open	close	adjclose	7_day_SMA	month	dayofyear	adjclose_lag_1	adjclose_lag_2	adjclose_lag_3	adjclose
11	2019-01-17	263.920013	259.959991	260.010010	262.959991	254.421967	251.215297	1	17	252.506241	251.896729	249.042496	250.
12	2019-01-18	266.980011	263.000000	264.980011	266.459991	257.808319	252.388772	1	18	254.421967	252.506241	251.896729	249.
13	2019-01-22	265.059998	261.059998	264.820007	262.859985	254.325195	252.938880	1	22	257.808319	254.421967	252.506241	251.
14	2019-01-23	264.790009	260.660004	264.010010	263.410004	254.857376	253.551189	1	23	254.325195	257.808319	254.421967	252.
15	2019-01-24	264.200012	262.079987	263.209991	263.549988	254.992783	254.401230	1	24	254.857376	254.325195	257.808319	254.

The next step is to normalize the dataset. The stock prices after normalizing would fall in an interval between 0 and 1. The normalization rule is as follows:

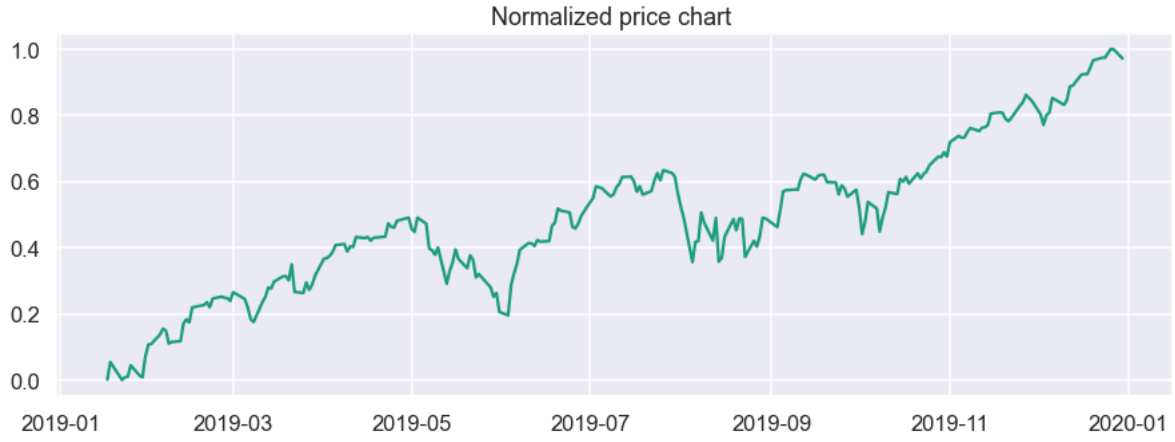
$$y = \frac{x - min}{max - min}$$

Where:
Y: normalized time-series value
X: the price from the input time series
Min: the minimum value of the time series
Max: the maximum value of the time series

This is my dataset after normalizing:

	date	adjclose_norm	high_norm	low_norm	open_norm	adjclose_lag_1_norm	adjclose_lag_2_norm	adjclose_lag_3_norm	adjclose_lag_4_norm	adjclose_la
11	2019-01-17	0.001511	0.001501	0.000000	0.000000	0.000000	0.000000	0.000000	0.022603	
12	2019-01-18	0.054385	0.052527	0.048781	0.077985	0.029086	0.009169	0.042196	0.000000	
13	2019-01-22	0.000000	0.020510	0.017651	0.075475	0.080499	0.037988	0.051206	0.042202	
14	2019-01-23	0.008309	0.016008	0.011233	0.062765	0.027616	0.088930	0.079528	0.051214	
15	2019-01-24	0.010424	0.006170	0.034018	0.050212	0.035696	0.036532	0.129590	0.079539	

This is adjusted close price after the normalization. There is no significant difference compared with the pre-normalized adjusted close price.



Implementation
First, I determined training, validation, and testing data sets. The training data set includes all data points from 2019-01-02 to 2019-06-30. The validation data set includes all data points from 2019-07-01 to 2019-09-30. The testing set includes all data points from 2019-10-01 to the end of the year.

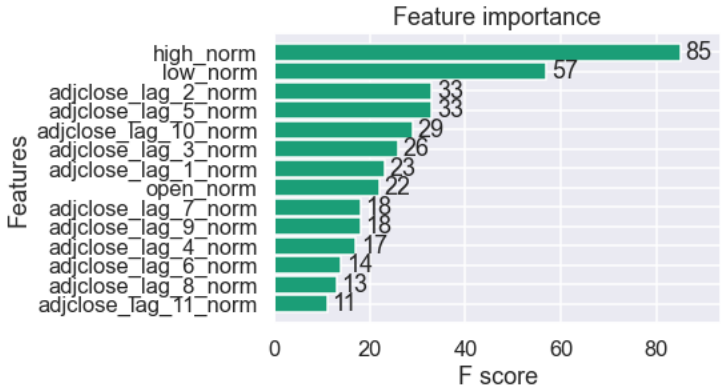
After splitting the data sets, I assigned X_train, y_train, X_val, y_val, X_test, and y_test. The dependent variables are the adjusted close prices split by the date milestones mentioned above. The independent variables are all remaining features split by the date milestones mentioned.

I passed the training data set into the XGBoost model. This is my model with associated parameters.

```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.300000012, max_delta_step=0, max_depth=6,
             min_child_weight=1, missing=nan, monotone_constraints='()',
             n_estimators=1000, n_jobs=0, num_parallel_tree=1, random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
             tree_method='exact', validate_parameters=1, verbosity=None)
```

The booster is a tree, in which the model consists of an ensemble of trees. The learning rate is 0.3. The learning rate is the shrinkage at every step, which ideally should be as low as possible. Increasing the learning rate makes the computation faster; however, it might potentially make the model not reach the best optimum. The model selects gamma equal to 0. Gamma is the minimum loss reduction to make a further partition on a leaf node of the tree. Larger gamma means that the algorithm becomes more conservative. Max depth is chosen to be 6. Max depth is the maximum depth of a tree. Increasing max depth makes the model more complex and more likely to overfit. The selected min_child_weight is 1. Min_child_weight is the minimum sum of instance weight needed in a child. Similar to gamma, larger min_child_weight means the algorithm is more conservative. Lambda is the L2 regularization terms on weights. Increasing lambda makes the model more conservative.

After fitting the model, I visualized the feature importance graph. Among the chosen features, it is unsurprising that high and low are the most important ones. Interestingly, among all lags, the 2nd and 5th lags are the most important ones.



Refinement

I used Grid Search to do hyperparameter tuning. The number of estimators is being chosen between 100, 200, 300, and 400. The learning rate choices are 0.001, 0.005, 0.01, and 0.05. The available max depth options are 8, 10, 12, and 15. The range of gamma is 0.001, 0.005, 0.01, and 0.02.

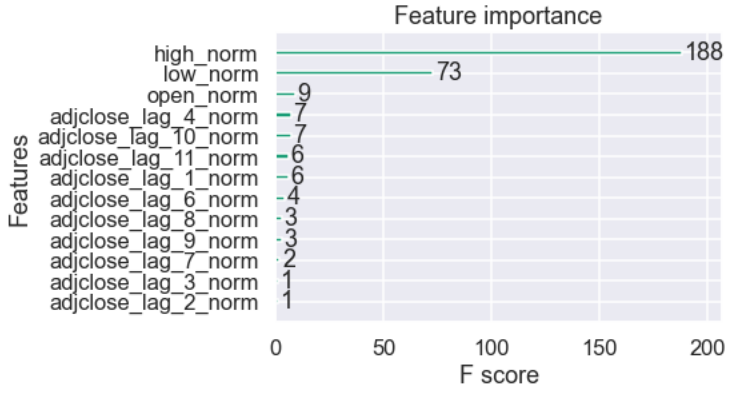
After fitting the parameters into Grid Search CV, the chosen gamma value is 0.001. The learning rate is 0.05. The max depth is 8. The number of estimators is 400. The random state is 42.

```
Best params: {'gamma': 0.001, 'learning_rate': 0.05, 'max_depth': 8, 'n
_estimators': 400, 'random_state': 42}
Best validation score = 0.3473118494871631
CPU times: user 5min 52s, sys: 5.28 s, total: 5min 57s
Wall time: 1min 40s
```

I fitted the new parameters into the model.

```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0.001, gpu_id=
-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.05, max_delta_step=0, max_depth=8,
             min_child_weight=1, missing=nan, monotone_constraints='()',
             n_estimators=400, n_jobs=0, num_parallel_tree=1, random_stat
e=42,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
             tree_method='exact', validate_parameters=1, verbosity=None)
```

I find the feature importance of the features after tuning the parameters. Only high and low prices become significantly important. Other variables have quite a low importance level.



RESULTS

Model Evaluation and Validation

The RMSE value of the first model is 0.28. However, the RMSE value for the tuned model is 0.3, which is higher. A potential reason is that the tuned parameters overfit the model, making the testing RMSE higher. A learning point is that tuning a model does not always improve the performance of the model. Tuning a model can lead to overfitting, which reduces the model performance.

Justification

The XGBoost algorithm returns better results compared with the simple moving average benchmarking model. The RMSE value of the SMA model is 2.76, while the RMSE value of the XGBoost model is 0.28. That difference in RMSE value is significant, meaning that there is a noticeable improvement from the benchmarking model to the chosen model.

CONCLUSION

I applied the XGBoost algorithm to the S&P 500 Index daily stock prices from 2019-02-01 to 2019-12-31. I applied three data augmentation methods, which are creating time-series related features, adding lag features, and normalizing the features. The XGBoost algorithm shows better performance in terms of RMSE compared with the benchmarking model, which is a simple moving average model. However, tuning the XGBoost model does not improve model performance. It reduces model performance by increasing test RMSE. Potential ideas to explore in the future include trying other machine learning methods (reinforcement learning and deep learning) and turning the problem statement into a binary one (predicting whether stock prices would go up or down) instead of predicting the exact values of the stock prices.

REFERENCES

- Bhaskar Sundaram, R., 2020. An End-To-End Guide To Understand The Math Behind Xgboost. [online] Analytics Vidhya. Available at: <http://An End-to-End Guide to Understand the Math behind XGBoost> [Accessed 14 November 2020].
- Daumé III, H., 2019. A Course In Machine Learning.
- Gidofalvi, Gyozo. Using News Articles to Predict Stock Price Movements. Department of Computer Science and Engineering, University of California, San Diego. 2001.
- Medium. 2020. Stock Price Prediction: Xgboost. [online] Available at: <https://medium.com/swlh/stock-price-prediction-xgboost-1fce6cbd24d7> [Accessed 15 November 2020].
- Predicting stock price using fuzzy grey prediction system Y.-F. Wang Department of Information Management, Chang Gung Institute of Nursing, Kwei-Shan, Tao-Yuan, Taiwan
- Roondiwala, M., Patel, H. and Varma, S., 2020. Predicting Stock Prices Using LSTM. International Journal of Science and Research (IJSR), 6(4), pp.1754-1756.
- Xgboost.readthedocs.io. 2020. Xgboost Parameters — Xgboost 1.3.0-SNAPSHOT Documentation. [online] Available at: <https://xgboost.readthedocs.io/en/latest/parameter.html> [Accessed 15 November 2020].