Data Scientist Nanodegree

Capstone Project

Yuyuan Lin
June 8, 2018

I. Definition

Project Overview

Many factors influence the stock price. As a result, the stock price is very volatile. Other than some professional traders or the ones with inside information, it is difficult to make profit from the stock investment. However, there are some patterns or trends in the historical stock price data. Prediction of stock price by analyzing the patterns or trends has shown some success and is widely used for professional traders. (refs 1-5) The analysis and prediction are facilitated with the advancement of machine learning techniques. (refs 6-10) In this study, I developed the machine learning algorithms to predict the S&P500 index. The developed algorithms were then used to predict the future S&P500 stock index. The machine learning algorithms include XGBoost, support vector machine, and neural network.

Problem Statement

The S&P500 index is reflects the overall stock market performance of the 500 large companies. The price of a stock is affected by many factors. Some of them are related to the financial performance or technological progresses of the company. Other factors such as interest rates, electron results, world events, rumors could also influence the stock price. (refs 11-12) In practice, the prediction of stock price by analyzing the company's revenue, profit, growth rate, and other financial situations is called fundamental analysis. The prediction of stock price based on the past pattern of a company's stock price is called technical analysis. (refs 13-15) The technical analysis demonstrated partial success as the past pattern of the stock price has already reflects all the factors that can influence the price. More recently, hybrid algorithms which combines the technical analysis and market sentiment data have been developed using machine learning techniques. (refs 18-20) For the scope of this study, I will focus on the technical analysis using machine learning techniques. The machine learning algorithms include XGBoost, support vector machine, and neural network.

Metrics

The predicted index of each algorithm is compared to the actual values. The performance of each machine learning technique will be quantified using Mean Squared Error (MSE) compared to the actual index. The MSE is defined in equation 1.

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

Since I will predict the stock price as a regression problem, using MSE is one of the most common choice. A better algorithm results in a smaller and MSE.

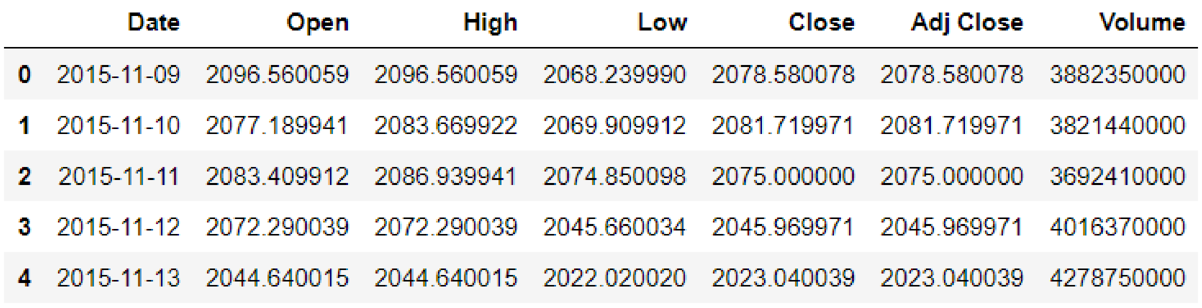
Due to the square nature in the MSE, this metric is more sensitive to outliners. It would also be good to use another metric, Mean Absolute Error (MAE) for a cross check. The MAE is defined in equation 2.

$$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n}\left|Y_i - \hat{Y}_i\right|^2 \qquad \text{Equation 1}$$

II. Analysis

Data Exploration

The historical data from 2015-11-09 to 2018-11-09 of S&P500 index was downloaded as a .csv file from Yahoo Finance. The first 5 rows of the index is shown in table 1. On each date, the index has Open, High, Low, Close, Adj Close, and Volume entries. The data is quite clean without NaN values. For simplicity, I only have predicted the Close index. Furthermore, I noticed that the values at Close and Adj Close columns are identical. So I ignored the Adj Close column in the analysis.



	Date	Open	High	Low	Close	Adj Close	Volume
0	2015-11-09	2096.560059	2096.560059	2068.239990	2078.580078	2078.580078	3882350000
1	2015-11-10	2077.189941	2083.669922	2069.909912	2081.719971	2081.719971	3821440000
2	2015-11-11	2083.409912	2086.939941	2074.850098	2075.000000	2075.000000	3692410000
3	2015-11-12	2072.290039	2072.290039	2045.660034	2045.969971	2045.969971	4016370000
4	2015-11-13	2044.640015	2044.640015	2022.020020	2023.040039	2023.040039	4278750000

Table 1

Fig. 1 shows the Close index with respect to the days passed since the starting date (2015-11-09). We can see a general upward trend with ups and downs over certain periods. The max index is 2930.75 and the minimum index is 1829.08.

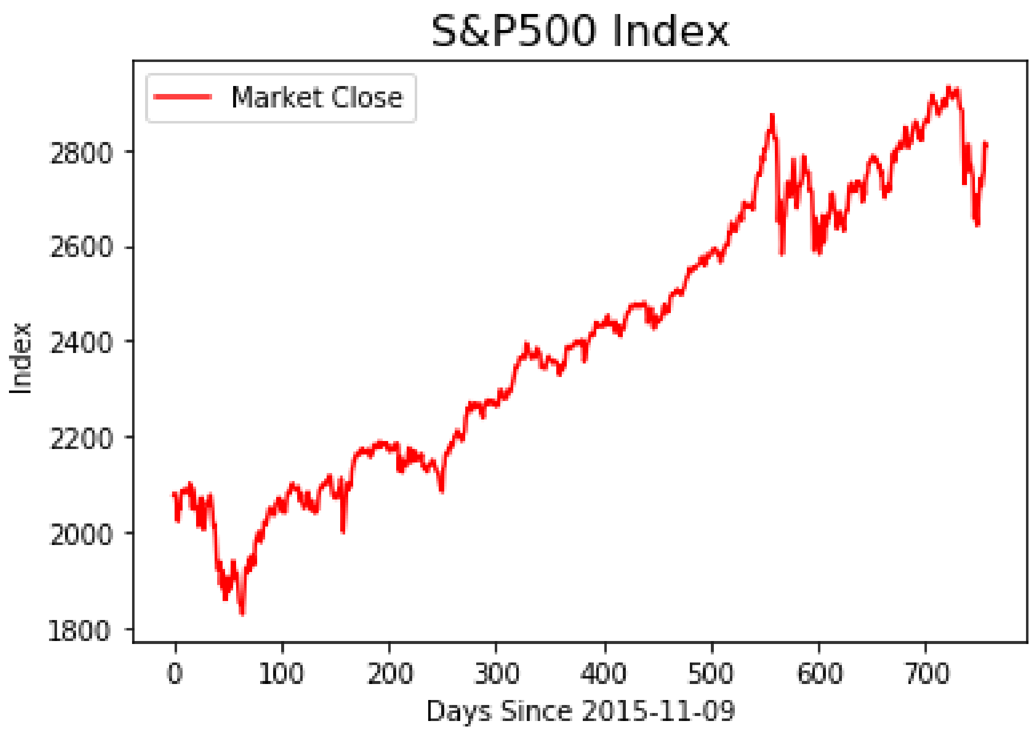


Figure 1

Algorithms and Techniques

The machine learning algorithms used in this study include XGBoost, support vector machine, and neural network. They are all popular algorithms nowadays for the prediction of stock price. Since the prediction of the exact index value is needed, I have used the regression version of each algorithm.

XGBoost

Boosting is an ensemble method where new predictors are added to the existing models sequentially until no improvement can be made. Gradient Boosting is an approach that tries to fit the new predictors to the residual errors made by the previous predictors. XGBoost is an implementation of Gradient Boosting. XGBoost has earned popularity in many data science competition, including stock price prediction, for its speed and performance. XGBoost has many hyperparameters to set. The hyperparameters can be divided into 3 categories:

- general parameters
    - booster [default=gbtree]
    - silent [default=0]
    - nthread [default to maximum number of threads available if not set]
- booster parameters
    - eta [default=0.3, alias: learning_rate]
    - min_child_weight [default=1]
    - max_depth [default=6]
    - gamma [default=0]
    - max_delta_step [default=0]
    - subsample [default=1]
    - colsample_bytree [default=1]
    - colsample_bylevel [default=1]
    - lambda [default=1]
    - alpha [default=0]
    - scale_pos_weight [default=1]
- learning task parameters
    - objective [default=reg:linear]
    - eval_metric [default according to objective]
    - seed [default=0]

I used the default general parameters and learning task parameters, tuned a few booster parameters including ‘eta’, 'min_child_weight' , 'max_depth', 'subsample', 'colsample_bytree' and ‘learning_rate’.

SVR

Support Vector Machine (SVM) is a very powerful and versatile machine learning algorithm. Its regression version (SVR) has been popular for a variety of prediction tasks including the stock price. It has hyperparamters:

- kernel [default='rbf']
- degree [default=3]
- gamma [default='auto']
- coef0 [default=0.0]
- tol [default=1e-3]
- C [default=1.0]
- Epsilon [default=0.1]
- Shrinking [default=True]
- cache_size [default = 200]
- verbose [default = False]
- max_iter [default=-1]

I tuned the kernel and C hyperparameters in the validation process and used the default values for the other hyperparameters.

FNN

Artificial neural network is a popular machine learning technique inspired by the biological neural networks. It has been widely used in computer vision, speech recognition, social network filtering, stock price prediction and others. Artificial neural network gives us many freedoms in constructing architectures, which results in many different neural network architectures. For the scope of this study, I only employed architecture with the fully-connected layers and fixed number of layers. The hyperparameters tweaked in this study are epochs and batch_size. Note that since it is a regression problem, the activation function of the output layer is set to linear. The FNN model is set:

```
# define the FNN model
def create_model():
    NN_model = Sequential()

# The Input Layer :
    NN_model.add(Dense(128, kernel_initializer='normal',input_dim = X_trainFNN.shape[1], activation='relu'))

# The Hidden Layers :
    NN_model.add(Dense(256, kernel_initializer='normal',activation='relu'))
    NN_model.add(Dense(256, kernel_initializer='normal',activation='relu'))
    NN_model.add(Dense(256, kernel_initializer='normal',activation='relu'))

# The Output Layer :
    NN_model.add(Dense(1, kernel_initializer='normal',activation='linear'))

# Compile the network :
    NN_model.compile(loss='mean_squared_error', optimizer='adam', metrics=['mean_squared_error'])
    return NN_model
```

The schematic of the above FNN construction is shown in Fig. 2.

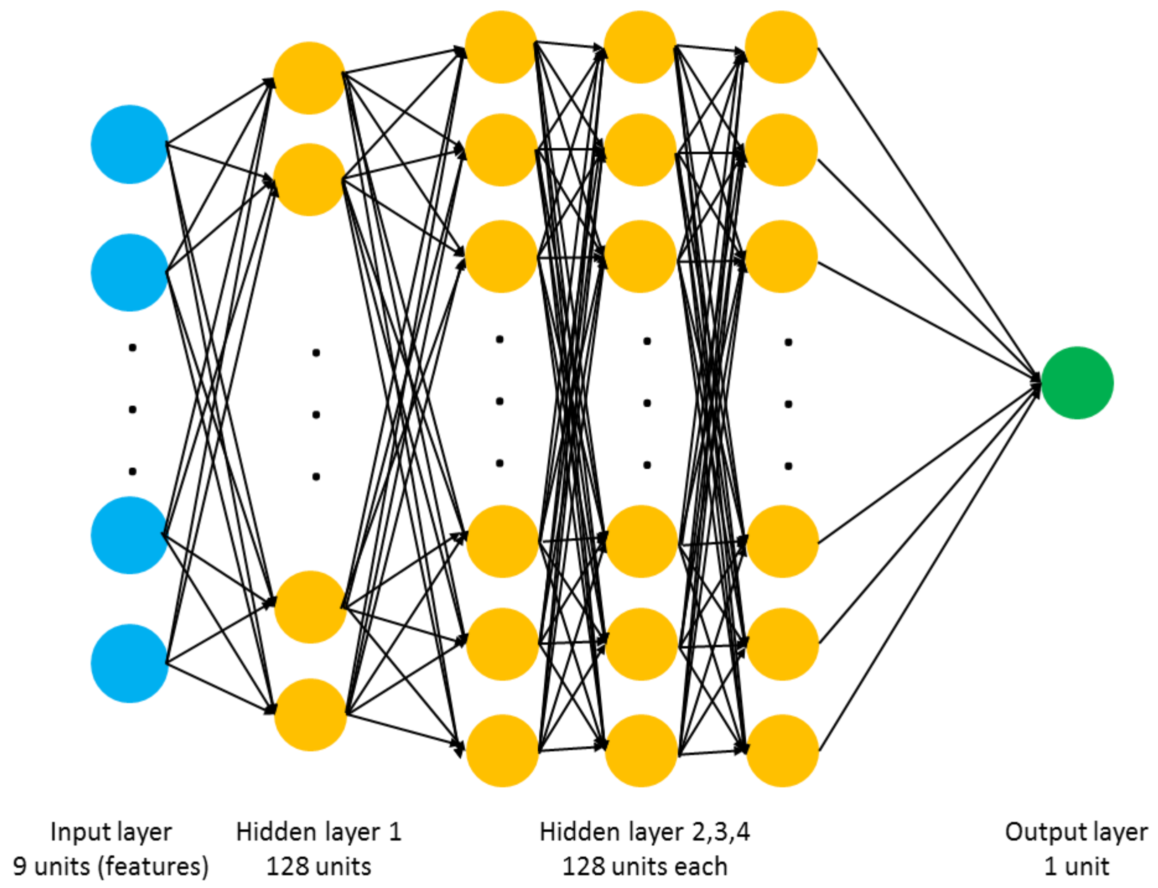


Figure 2

For the above three algorithms, the hyperparameters were tweaked in the validation process using grid search method. As the dataset of S&P 500 index is a time-series data, the data split for training and validation should not be assigned randomly. Instead, Time SeriesSplit() function was used for the cross-validation process in order to limit the look--ahead bias in the time-series data.

Benchmark

I used a persistence model for the benchmark. The persistence model does nothing except shifting the index value by 1 day. No training of the input data is required. For example, today's index value is 2000. Using the persistence model, I predict tomorrow's index value is 2000. As a result, the pattern of predicted index by the persistence model is identical to the real index, except a shift in the X-axis, as shown in table 2 and Fig. 3.

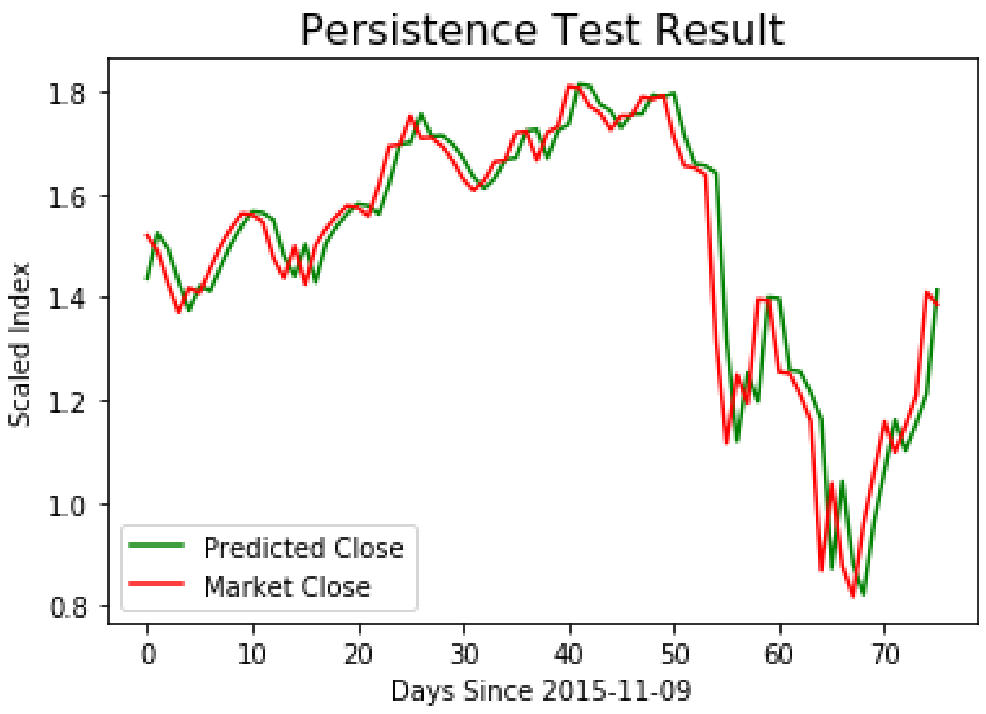


Figure 3

	Close Shift	Close
1	2078.580078	2081.719971
2	2081.719971	2075.000000
3	2075.000000	2045.969971
4	2045.969971	2023.040039
5	2023.040039	2053.189941

Table 2

In reality, using the persistence model to predict the stock price is naïve as it follows the trend but is delayed by 1 day (or a certain time). It the stock price is not very volatile, this investment strategy is OK. However, for those stocks with rapid ups and downs in a daily basis, the persistence strategy should be abandoned. If the performance of any machine learning algorithms worse than the persistence model should be further improved or abandoned.

III. Methodology

Data Preprocessing

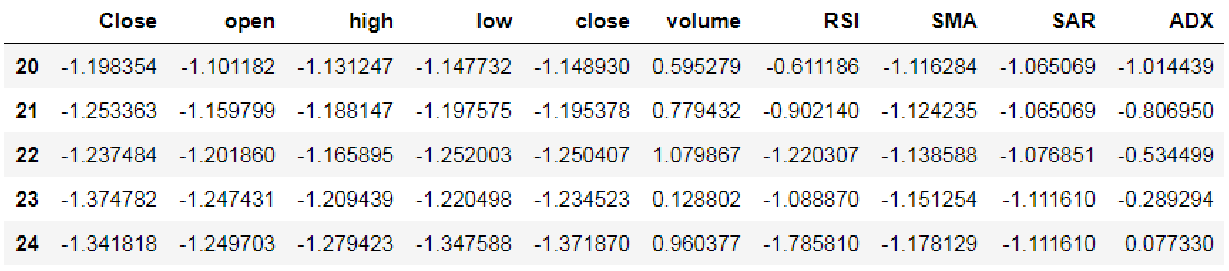
The first thing to do is to shift the index in the Open, High, Low, Close and Volume columns by a day and save to new columns named open, high, low, close and volume respectively. Furthermore, I dropped the columns of Open, High, Low and Volume for the later analysis. This is similar to the persistence model. The assumption is that the prediction of next day's index is based on available information on the day before the next day. In addition, I added more features to the original data. The features are some technical indicators that were often used by the profession traders. The 4 indicators used are relative strength index (RSI), simple moving average (SMA), parabolic stop and reverse (SAR), and average directional movement Index (ADX). The definitions of the 4 indicators are shown in table 3.

Indicators	Description
RSI	RSI = 100 – [100 / (1 + (Average of Upward Price Change / Average of Downward Price Change))]
SMA	$\frac{1}{n}\sum_{t-(n-1)}^{t} P_i$,where P is the stock price, n is the time period
SAR	Please refer to https://stockcharts.com/school/doku.php?id=chart_school:technical_indicators:parabolic_sar
ADX	https://en.wikipedia.org/wiki/Average_directional_movement_index

Table 3

The new input data has NaN values after the calculation of indicators. I simply dropped the entries with NaN values.

The next step is to scale the data of each feature (column) by the StandardScaler method, which standardizes features by subtracting the mean and scaling to unit variance. This is to make sure the algorithms learn from each feature. Table 4 shows an example of the processed data used as the input for all three machine learning techniques.



	Close	open	high	low	close	volume	RSI	SMA	SAR	ADX
20	-1.198354	-1.101182	-1.131247	-1.147732	-1.148930	0.595279	-0.611186	-1.116284	-1.065069	-1.014439
21	-1.253363	-1.159799	-1.188147	-1.197575	-1.195378	0.779432	-0.902140	-1.124235	-1.065069	-0.806950
22	-1.237484	-1.201860	-1.165895	-1.252003	-1.250407	1.079867	-1.220307	-1.138588	-1.076851	-0.534499
23	-1.374782	-1.247431	-1.209439	-1.220498	-1.234523	0.128802	-1.088870	-1.151254	-1.111610	-0.289294
24	-1.341818	-1.249703	-1.279423	-1.347588	-1.371870	0.960377	-1.785810	-1.178129	-1.111610	0.077330

Table 4

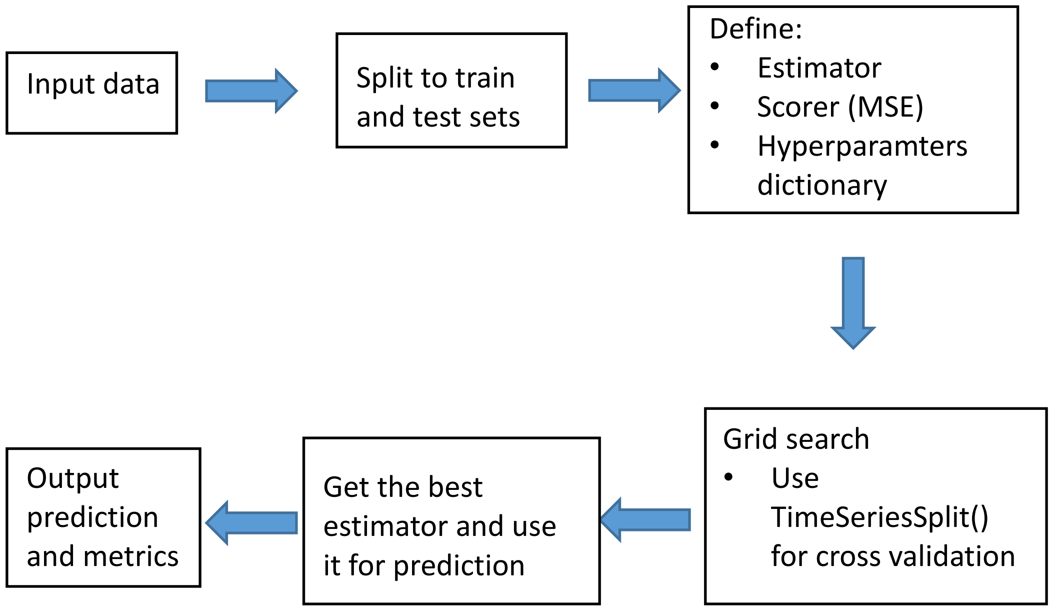
Implementation

For the implementation of all three algorithms, I used the same type of flow, as illustrated Fig. 4. The 90% of the input data was used to training and 10% of them was used for testing. The 'Close' column is the "y", and the other columns are the "X". The mean_squared_error scorer was used in each validation process. The hyperparameters used for tuning was listed in the algorithms and techniques section. A grid search was used each machine learning algorithm to find the best combination of the hyperparameters. As the input is a time-series data, the TimeSeriesSplit() method was used for cross-validation. 5 splits were used for the cross-validation process. The best estimator for machine learning algorithm was used to predict the index. The prediction results were evaluated by the MSE compared to the real scaled Close index. A curve was also plotted to show the qualitative difference between the predicted and real index for each algorithm.

Using the grid search technique takes longer time to obtain the output and is more likely to introduce some bugs. What I did is to use fewer grids at the beginning. At the later stage, I added more parameters for the optimization to sweep. It is also a good idea to have fewer training data first at the early stage of implementation. As a result, I was able to notice any bugs in the implementation without waiting a long time for the computation to finish.

It is convenient that the TimeSeriesSplit() method was available. I only need to pass it to the grid search method.

With many variables existing in a single ipython file, the code implementation is less readable and more subjected to errors. I tried to use different variable names for each machine learning algorithm. If the project is for my own use, I would separate each algorithm with a different ipython file. Or I will try to ensemble them using the scipy.optimize.minimize method for the next version of the implementation. For the submitted ipython file, I removed the implementation of metrics using MAE, as the implementation is very similar to the metric using MSE.


Input data
Split to train and test sets
Define:
• Estimator
• Scorer (MSE)
• Hyperparamters dictionary
Grid search
• Use TimeSeriesSplit() for cross validation
Get the best estimator and use it for prediction
Output prediction and metrics

.

Figure 4

Refinement

The refinement process was completed automatically with the grid search method. The performance of each hyperparameter combination were evaluated by the MSE scorer and printed out.

- For SVR,
    - Using linear kernel results in better performance than using the rbf kernel. The mean MSE is ~ 0.0046 and ~0.14 using linear and rbf kernel respectively calculated in the cross-validation process.

    - The MAE scorer results in the same best combination of hyperparameters.
    - The C value, one the other hand, does not impact the training performance in this study.
- For FNN
    - The best combination is 50 epochs and 64 batch_size. The cross-validation mean MSE is ~0.0294. The second best combination is to use 100 epochs and 64 batch_size. The mean MSE is ~0.00379 in the cross-validation process.
    - The MAE scorer selected 100 epochs and a batch size of 32.
- For XGBoost
    - For the about 50 combinations of hyperparameters tried, the results are similar with each combination. The best combination is 0.3 learning_rate, 0.8 colsample_bytree, 1 min_child_weight, 0.5 subsample, 5 max_depth. The mean MSE is 0.1425 in the cross-validation process.
    - The MAE scorer selected the same combination of hyperparameters for the best performance.

IV. Results

Model Evaluation and Validation

The final models were selected by the grid search cross validation process.

The final SVR model used is:

SVR(C=5, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma='scale', kernel='linear', max_iter=-1, shrinking=True, tol=0.001, verbose=False)

Using this model, the MSE for the test set is 0.0076. The MAE for the test set is 0.0600.

The final FNN model used is:

1 input layer, 3 hidden layer, 1 output layer, 'normal' kernel initializer, 'reLu' activation function except the output layer used a linear activation for the regression. The optimizer was set as "adam'. 50 epochs and a batch_size of 64 were used. Both loss and metric function used MSE. The final MSE for the test set is 0.0099, while the MAE is 0.0825.

The final XGBoost model used is:

XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1, colsample_bytree=0.8, gamma=0, learning_rate=0.3, max_delta_step=0, max_depth=5, min_child_weight=1, missing=None, n_estimators=500,n_jobs=1, nthread=None, objective='reg:linear', random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None, silent=True, subsample=0.5) Using this model, the MSE for the test set is 0.0210. The MAE for the test set is 0.2421.

Justification

Using the benchmark persistence model, the MSE and MAE for the test set is 0.0074 and 0.0586 respectively. As a result, the SVR, FNN, and XGBoost models were all outperformed by the benchmark model. The SVR model results in almost identical performance to the benchmark model. I have tried to remove the 'close' column, in order to help avoid reaching the local minimum as reached by the persistence model. And then the test MSE (0.0072) using the SVR was a little less than the persistence model. Table 4 summarizes the performance of each machine learning algorithm in the test set.

Algorithm	MSE	MAE
Persistence	0.0074	0.0586
SVR	0.0076	0.0600
FNN	0.0099	0.0825
XGBoost	0.0833	0.2421

Table 5

V. Conclusion

Free-Form Visualization

The comparison of predicted close index and market close index using SVR, FNN, and XGBoost is shown in Figs 5, 6, and 7 respectively. The prediction made by SVR is almost identical to the persistence model, which is almost a shift of the index curve along the X-axis by a day. On the other hand, obvious mismatch is observed between the predicted index by the XGBoost model and the real index. The performance of FNN is in between of SVR and XGBoost.

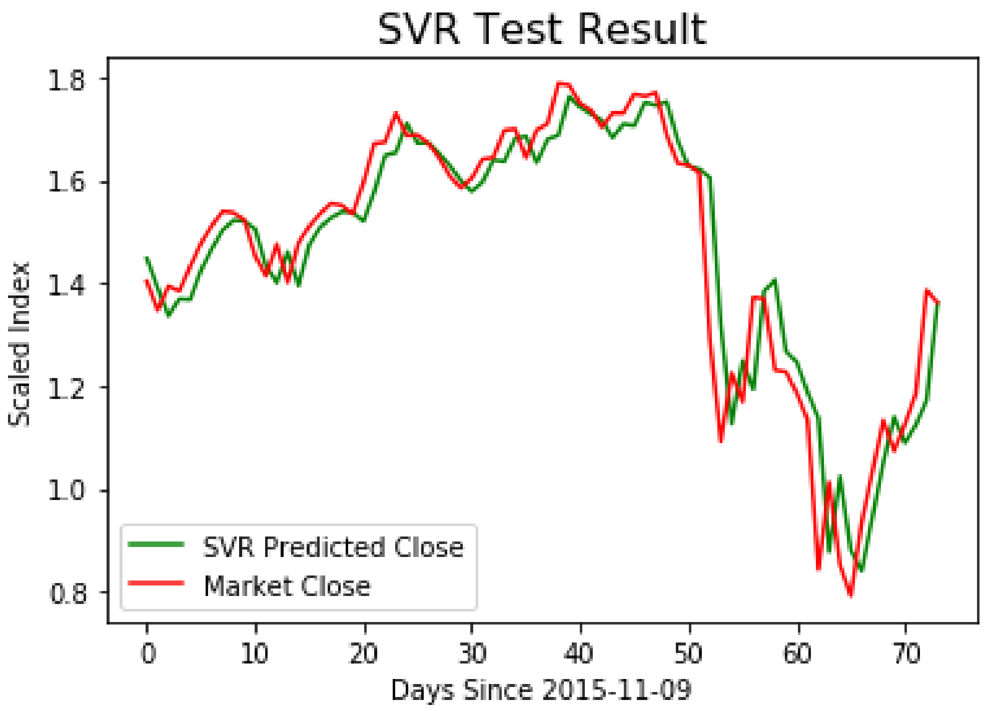


Figure 5

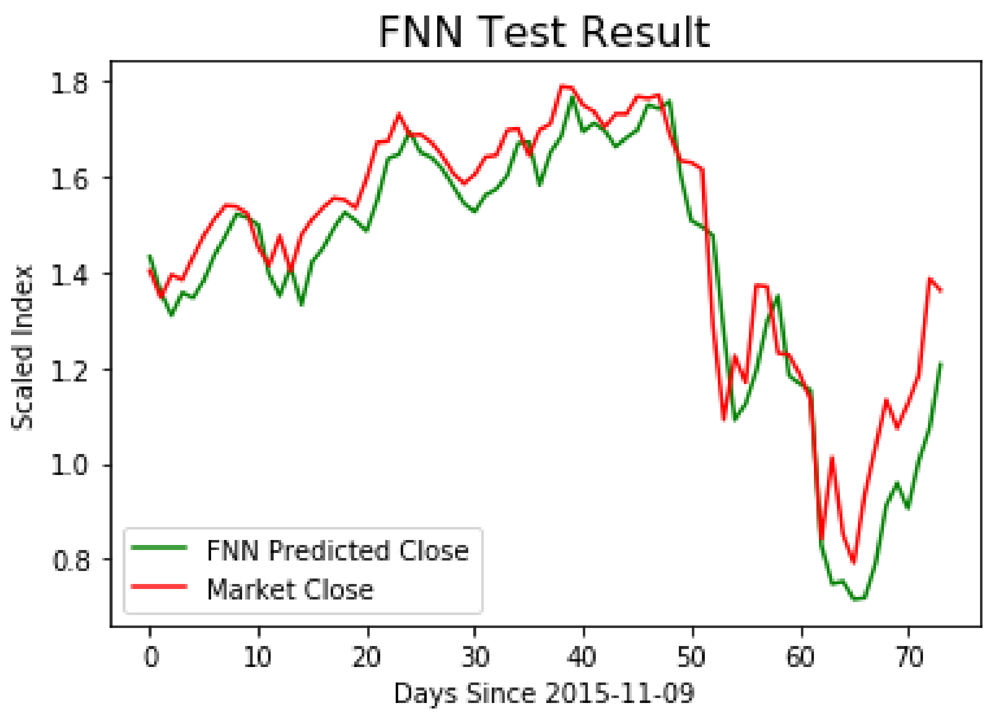


Figure 6

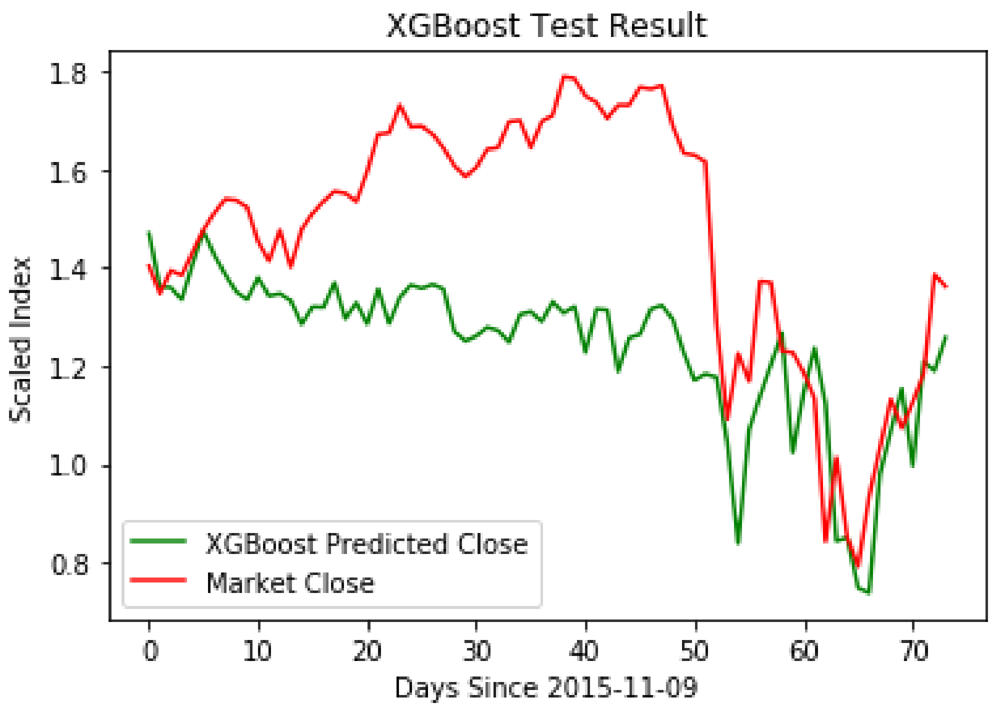


Figure 7

To evaluate the robustness of the final models, I checked predicted the indices of NASDAQ, and Dow Jones. Since S&P500 index is correlated to NASDAQ and Dow Jones, a robust model should predicted NASDAQ and Dow Jones indices almost equally well. The MSE for the prediction of each index is shown in table 6. The comparison between the predicted results and market results are shown in Fig.8. It is obvious that the SVR model is robust, which results in the similar prediction as the persistence model did. The FNN model is doing fairly well. However, the prediction results are subjected to change a little even I run the training and prediction again on the same data set. With so many units to fit for the FNN, I consider it is not bad. But the FNN might overkill this study. It is difficult to evaluate the XGBoost model, as the performance is not good for all 3 indices. Further improvements are needed for this model.

Algorithm	MSE (S&P500)	MSE (NASDAQ)	MSE (Dow Jones)
SVR	0.0076	0.0077	0.0054
FNN	0.0099	0.0113	0.0103
XGBoost	0.0833	0.1391	0.0451

Table 6

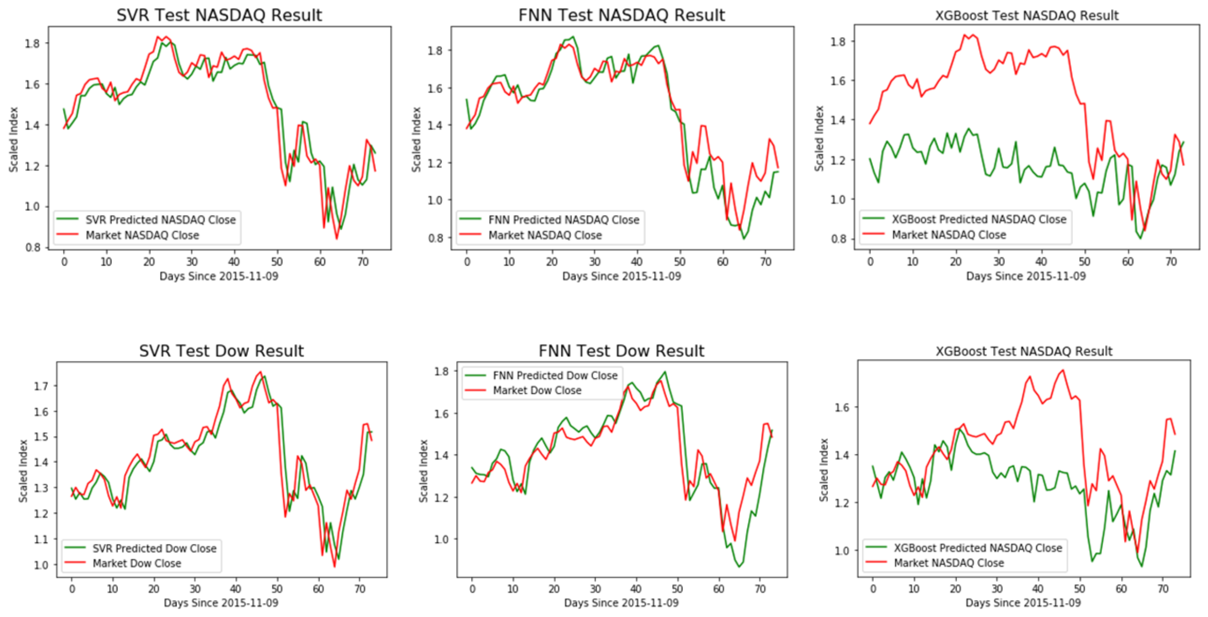


Figure 8

Reflection

The purpose of this study is to use machine learning algorithm to predict the future (next day) S&P500 index, assuming the index can be predicted based on the historical data. The machine learning algorithms used in this study are SVR, FNN, and XGBoost. A few extra features (technical indicators) were added in order to develop a more reliable index predictor. A 3-year historical data obtained from Yahoo Finance was split to training and testing set in a 9:1 ratio. The algorithms were trained using the trained data. Some of the hyperparamters were determined in the cross-validation process. A grid search method was used in the cross-validation process, and the training and validation split was determined by a TimeSeriesSplit() function to avoid look-ahead bias. Some other parameters were chosen as default. The best model determined by the cross-validation process was used to predict the index. The evaluation of each algorithm was measured by the MSE on the test data set.

The result is sort of disappointing as no advanced algorithms in this study, namely SVR, FNN, and XGBoost, can outperform the benchmark model (persistence model). The SVR model results in almost identical performance to the benchmark model. On the other hand, it is not that surprising that one cannot beat the market. In fact, most people earn less than the market return. (ref 16-18) The project is interesting as one can make some profit if a good model is developed. The most challenging thing is to decide which and how many technical indicators to use. Fortunately the library Ta-lib is available for the implementation of the technical indicators.

Improvement

In this study, only technical indicator and historical values were used for the index prediction. As the stock price is also impacted by other factors such as macro-economics, politics, natural and man-made disasters, market psychology, it is reasonable to consider adding some market sentiment indicators for the input. (refs 18-20)

In addition, I would like to try other neural network architectures, such as LSTM and Q-learning. Those architectures have demonstrated reasonable success in the prediction of stock market. (refs 21-23). The XGBoost model also needs a significant improvement. I will learn some skills from the Kaggle winners for this model.

Last but not least, I will make more beautiful and professional plots using other visualization libraries.

Acknowledgment

For the implementation, I have learned a lot from

Jason Brownlee’s posts on https://machinelearningmastery.com/

Posts on www.quantinsti.com/

Reference

1. https://www.investopedia.com/terms/t/technicalanalysis.asp
2. https://en.wikipedia.org/wiki/Technical_analysis
3. https://en.wikipedia.org/wiki/Stock_market_prediction
4. https://www.quora.com/Is-it-possible-to-predict-stock-market-trends-from-historical-data-points-If-yes-what-kind-of-algorithms-are-in-use
5. https://www.moneycrashers.com/the-best-way-to-invest-fundamental-or-technical-analysis/
6. https://towardsdatascience.com/stock-prediction-in-python-b66555171a2
7. https://www.quora.com/Can-machine-learning-algorithms-models-predict-the-stock-prices-If-yes-which-are-the-best-machine-learning-algorithm-models-to-predict-the-stock-prices
8. https://epublications.bond.edu.au/infotech_pubs/110/
9. https://www.sciencedirect.com/science/article/abs/pii/S037872060100091X
10. https://ieeexplore.ieee.org/abstract/document/931880
11. https://www.getsmarteraboutmoney.ca/invest/investment-products/stocks/factors-that-can-affect-stock-prices/
12. https://www.investopedia.com/articles/basics/04/100804.asp
13. http://www.businessdictionary.com/article/1104/fundamental-analysis-vs-technical-analysis-d1412/
14. https://www.investopedia.com/university/technical/techanalysis2.asp
15. https://www.moneycrashers.com/the-best-way-to-invest-fundamental-or-technical-analysis/
16. https://www.investopedia.com/ask/answers/12/beating-the-market.asp
17. https://www.thesimpledollar.com/even-the-experts-cant-beat-the-market-why-would-you/
18. https://www.fool.com/investing/2017/09/23/warren-buffett-on-beating-the-stock-market.aspx
19. https://www.researchgate.net/publication/262155335_Machine_Learning_in_Prediction_of_Stock_Market_Indicators_Based_on_Historical_Data_and_Data_from_Twitter_Sentiment_Analysis
20. http://jonathankinlay.com/2016/11/trading-market-sentiment/
21. https://link.springer.com/chapter/10.1007/3-540-46146-9_16
22. https://github.com/filangel/qtrader
23. https://github.com/kh-kim/stock_market_reinforcement_learning