Predict the close price of BMW

Leyu Pan
Ningbo Xiaoshi High School
Ningbo, Zhejiang, China
xhd789@163.com

Abstract

Some stocks in the stock market are related to each other, especially those in the same field. In this paper, the stock prices of several motor corporations and three economic indices (S&P 500, CSI 300 and US Dollar Index) were perceived as features for predicting the close price of BMW. Models including XGBoost, Random Forest Regressor, Linear Model 'Ridge' and Linear Regression were applied to the data and measured by r squared score. I compared the performance of calculating geometric mean of predicted values with using a single model and found that geometric mean was closer to the real value. Through this project, I also considered that XGBoost seems to be better for forecasting long-term data. What's more, forecasting long-term data is more likely to yield more accurate results than short-term data.

1 Introduction

Stock price prediction is a project that combines finance and machine learning. Plenty of researches has been carried out and we can learn from these previous works that there are a lot of different prediction methods.

Rebwar M. Nabi, Soran Ab. M. Saeed and Habibollah Harron introduced a multiclass classification ensemble learning approach, which used Gradient Boosting Machine with Feature Engineering, and achieved a low MAPE (0.0406%) [1]. After comparing the predicted result of SVM and Linear Regression, Bhawna Panwar, Gaurav Dhuriya, Prashant Johri, Sudeept Singh Yadav and Nitin Gaur came to a conclusion that Linear Regression performs better than SVM in predicting stock price [2]. Ayodele A. Adebiyi, Aderemi O. Adewumi and Charles K. Ayo predicted the stock price based on the ARIMA model and considered that this model has great potential and competitiveness in forecasting short-term stock price data [3]. Based on Mixed model of ARIMA and XGBoost, Yan Wang and Yuankai Guo improved the predicting result of just use the ARIMA model or XGBoost model [4]. In addition to predicting specific prices, Indranil Ghosh and Tamal Datta Chaudhuri focused on the trend of stock price and forecasted it by regarding it as a binary classification (whether the trend would rise or fall) based on FEB-Stacking and FEB-DNN models [5].

These previous researches gave me some inspiration. In this work, I applied seven models respectively (XGBoost, Random Forest Regressor, Linear Model 'Ridge' and Linear Regression) to predict the closing price of BMW. And on this basis, the geometric mean of each predicted value was carried out to observe whether the result improved. Furthermore, I made a simple comparison on the prediction of XGBoost model with data of different lengths.

I uploaded my code to Github:

https://github.com/lorielp/final-project-BMW_Close-predict

Preprint. Under review.

2 Data Preparation

First of all, by downloading files from 'Kaggle' and using two Python modules ('pandas_datareader.data' and 'tushare'), I got the required data. Then I did some EDA and merged the processed data together. After comparing with other methods, 'Linear Interpolation', a 'scipy' method, was found to be a better strategy, so I applied it to handle the missing data after comparing with several strategies.

2.1 Obtain data

Kaggle Stock price of some car companies are available on 'Kaggle' at:

https://www.kaggle.com/meetnagadia/share-price-of-top-electric-car-company/download

Although the title says 'electric-car', after I checked, actually, it's not the stock price of a separate electric segment, but the stock price of companies with electric cars.

Then I took the historical stock price data of BMW, Volkswagen, Tesla, Nissan, Honda and Rolls-Royce from the downloaded file.

pandas_datareader.data The Python package 'pandas_datareader.data' is a tool to crawl financial data. Just take Toyota as an example:

```
import pandas_datareader.data as pdr
toyota=pdr.DataReader(name='7203.T',data_source='yahoo',start='2016-08
                                     -24',end='2021-08-23')
```

Through this method, stock price of other seven companies (Toyota Motor Corporation, Porsche Automobil Holding SE, Daimler AG, General Motors Company, Ford Motor Company, Peugeot Invest Société anonyme, Stellantis N.V.) and historical data of two economic indices (S&P 500, US Dollar Index) were successfully acquired.

tushare 'tushare' is similar to 'pandas_datareader.data', but it mainly focuses on the China stock market. So I used it to get 'CSI 300' data, which can present the overall performance of the A shares traded on Shanghai and Shenzhen Stock Exchange [6].

```
import tushare as ts
CSI300=ts.get_k_data('hs300',start='2016-08-24',end='2021-08-23')
```

2.2 Normalise data frame

In order to reduce the effect of future function, I extracted the opening price of the day and the other price of the previous day for stocks and indices other than BMW.

```
CSI300 = loadData('CSI300').iloc[:, :-1]
CSI300['High_yesterday'] = CSI300['High'].shift(1)
CSI300['Low_yesterday'] = CSI300['Low'].shift(1)
CSI300['Close_yesterday'] = CSI300['Close'].shift(1)
CSI300 = CSI300.drop(['High','Low','Close'],axis=1).iloc[1:]
CSI300
```

	Open	High_yesterday	Low_yesterday	Close_yesterday
Date				
2016-08-25	3314.08	3348.13	3323.73	3329.86
2016-08-26	3312.50	3315.08	3279.81	3308.97
2016-08-29	3306.57	3328.95	3301.74	3307.09
2016-08-30	3310.32	3315.08	3297.07	3307.78
2016-08-31	3310.57	3325.12	3304.60	3311.99
...	...	...	...	...
2021-08-17	4935.00	4973.38	4931.37	4941.07
2021-08-18	4838.47	4967.56	4823.39	4837.40
2021-08-19	4886.56	4909.54	4818.90	4894.24
2021-08-20	4817.49	4893.47	4839.57	4862.14
2021-08-23	4777.73	4835.15	4720.83	4769.27

1215 rows × 4 columns

As for the data frame of BMW, I kept the opening and closing prices of the day. And the highest and lowest prices were also shifted by one day. After that, I added the name of the data to each column and merged all the data frames together.

	bmw_Open	bmw_Close	bmw_High_yesterday	bmw_Low_yesterday	Volkswagen_Open	Volkswagen_High_yesterday	Volkswagen_Low_yesterday	Vol
Date								
2016-08-25	76.800003	76.220001	78.029999	76.690002	122.050003	123.900002	121.150002	
2016-08-26	76.099998	77.230003	76.839996	75.599998	120.699997	122.400002	119.449997	
2016-08-29	76.709999	76.879997	77.290001	75.820000	123.199997	124.250000	120.550003	
2016-08-30	77.440002	78.459999	77.230003	76.070000	123.250000	124.900002	121.849998	
2016-08-31	78.379997	78.010002	78.559998	77.110001	124.400002	125.250000	122.800003	
...	...	...	...	...	...	...	...	

2.3 Fill missing values

I found that among these rows, there existed some missing values. During the course, we have learned 'Simple Imputer', which can help us to deal with it. In my spare time, I read a jupyter notebook mentioned that 'interp1d' from 'scipy' also has the ability to handle missing values [7].

Therefore, the next step was to measure which filling strategy would be better for this project. I randomly selected 10 columns and made an empty data frame by using:

```
cols = DATA.sample(n=10, frac=None, replace=False, weights=None, random_state=42, axis=1).columns
errdf = pd.DataFrame(index=[
    'Simple Imputer', 'linear', 'nearest', 'zero', 'slinear', 'quadratic',
    'cubic', 'previous', 'next'],
                     columns=[cols])
errdf
```

	SP500_Open	CSI300_Low_yesterday	bmw_Open	peugeot_Open	Volkswagen_High_yesterday	General_M_Open	honda_Open	nissan_Open	toyota_Hi
Simple Imputer	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
linear	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
nearest	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
zero	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
slinear	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
quadratic	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
cubic	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
previous	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
next	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	

After that, I defined a function which will drop 20% value of each column at random, applied all the methods to fill NaN and then compared the mean square error between real value and filled value.

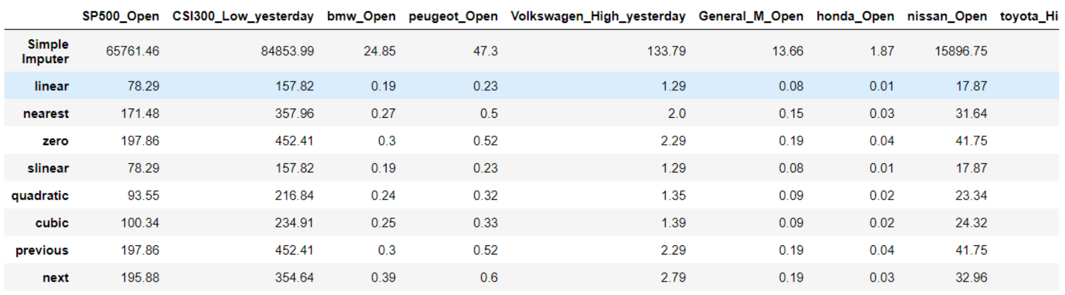
```
from scipy.interpolate import interp1d

def m_s_e(name):
    df_orig0 = pd.DataFrame(DATA[name].dropna())
    df0 = df_orig0.sample(n=int(df_orig0.shape[0] * 0.8),
                          frac=None,
                          replace=False,
                          weights=None,
                          random_state=41,
                          axis=0).rename(columns={name: 'value'})
    dfdf = mergeData([df_orig0, df0])
    df_orig = pd.DataFrame(dfdf[name]).rename(columns={name: 'value'})
    df = pd.DataFrame(dfdf['value'])

    # Simple Imputer
    df_imput = SimpleImputer().fit_transform(df)
    err_imp = np.round(mean_squared_error(df_orig['value'], df_imput),
                       2)
    errdf.loc['Simple Imputer', name] = err_imp

    # interp1d
    parameters = ['linear', 'nearest', 'zero', 'slinear', 'quadratic',
                  'cubic', 'previous', 'next']
    df['rownum'] = np.arange(df.shape[0])
    df_nona = df.dropna(subset=['value'])
    for param in parameters:
        f = interp1d(df_nona['rownum'], df_nona['value'], kind=param)
        df[param + '_fill'] = f(df['rownum'])
        error = np.round(
            mean_squared_error(df_orig['value'],df[param+'_fill']),2)
        errdf.loc[param, name] = error
    return errdf
```

The mean square error of each strategy for each column is shown in the data frame below:



	SP500_Open	CSI300_Low_yesterday	bmw_Open	peugeot_Open	Volkswagen_High_yesterday	General_M_Open	honda_Open	nissan_Open	toyota_Hi
Simple Imputer	65761.46	84853.99	24.85	47.3	133.79	13.66	1.87	15896.75	
linear	78.29	157.82	0.19	0.23	1.29	0.08	0.01	17.87	
nearest	171.48	357.96	0.27	0.5	2.0	0.15	0.03	31.64	
zero	197.86	452.41	0.3	0.52	2.29	0.19	0.04	41.75	
slinear	78.29	157.82	0.19	0.23	1.29	0.08	0.01	17.87	
quadratic	93.55	216.84	0.24	0.32	1.35	0.09	0.02	23.34	
cubic	100.34	234.91	0.25	0.33	1.39	0.09	0.02	24.32	
previous	197.86	452.41	0.3	0.52	2.29	0.19	0.04	41.75	
next	195.88	354.64	0.39	0.6	2.79	0.19	0.03	32.96	

With the help of 'for-loop' and '.idxmin()', it's significant that 'Linear Interpolation' performed the best for these randomly-chose columns. Therefore, I used this way to deal with all the columns.

```
for n in errdf.columns:
    print(errdf[n].astype(float).idxmin())
```

```
linear
linear
linear
linear
linear
linear
linear
linear
linear
linear
```

3 Predict test data

I took the close price of BMW as label y. And all the other columns (including ‘bmw_Open’, ‘bmw_High_yesterday’, ‘bmw_Low_yesterday’, the opening price of the day, the closing, highest and lowest price of the previous day for the stocks and economic indices other than BMW) were regarded as feature x.

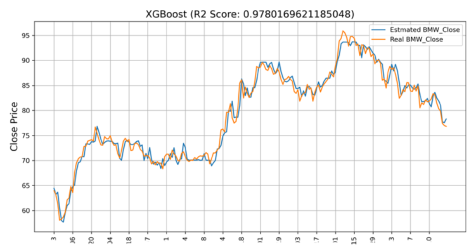
‘TimeSeriesSplit’ and ‘GridSearchCV’ are both Scikit-Learn methods. ‘TimeSeriesSplit’ is designed for time series train-test split and cross validation, instead of the general ‘train_test_split’ and cross validate. ‘GridSearchCV’ can help to find the better parameters for the models.

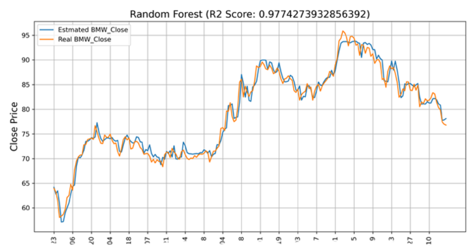
In my project, I set ‘n_splits’ of ‘TimeSeriesSplit’ as 5 and combined it with ‘GridSearchCV’, using XGBoost, Random Forest Regressor and linear model ‘Ridge’ to fit the x and y in order to determine the parameters for each model.

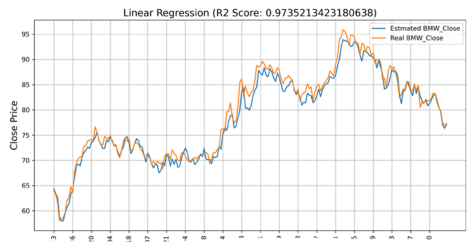
Since the last set of training set while using ‘TimeSeriesSplit’ (n_splits=5) was the first 1055 data, I divided training set and test set as follows:

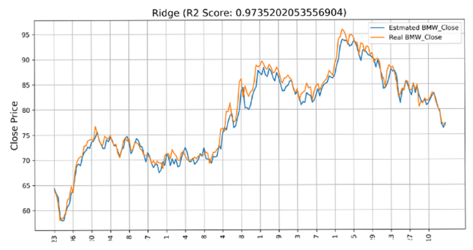
```
x_train, y_train = x[0:1055], y[0:1055]
x_test, y_test = x[1055:], y[1055:]
```

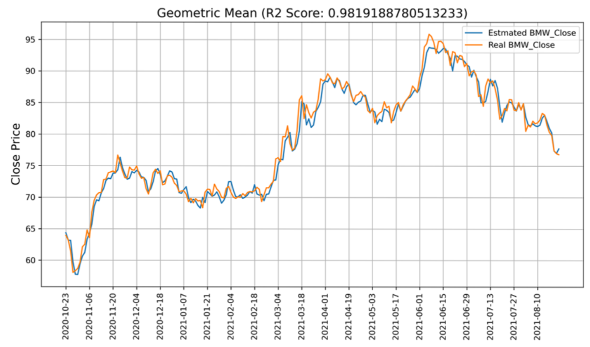
I trained the models above and Linear Regression model by using ‘x_train’ and ‘y_train’, and then predicted ‘y_test’ respectively. After that, I calculated the geometric mean of all the predicted values. The reason why I chose ‘r squared score’ to measure the accuracy is that it’s the most commonly used indicator to evaluate regression models.









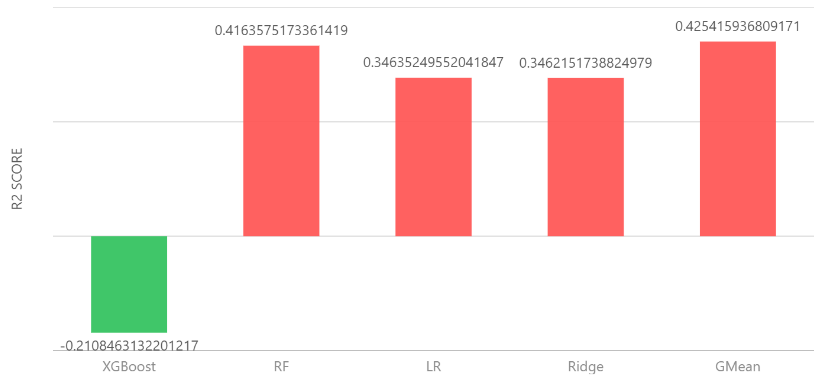


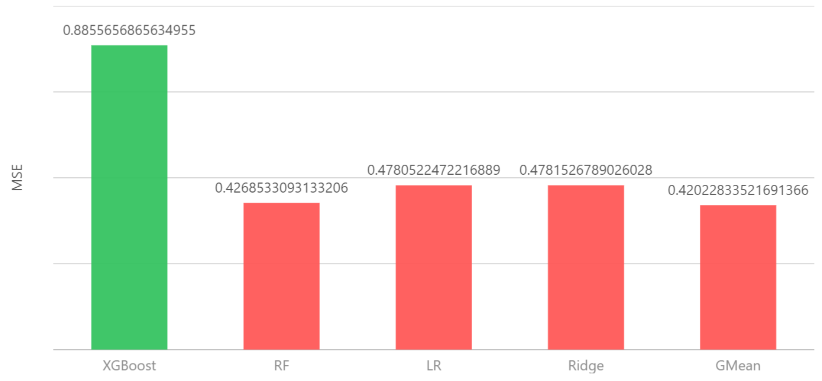
According the figures above, it’s clear that XGBoost model reached the highest score among the four models, followed by Random Forest Regressor, while linear models (Ridge and Linear Regression) had similar scores. And the geometric mean, compared to the original predicted values, showed a further improvement in the score.

4 Short-term Prediction

We can find that the test errors seemed to be quite satisfactory. Would similar result be achieved if the process introduced above had been implemented to predict a short-term data? Thus, I obtained the feature data from 2021-08-24 to 2021-09-07, which actually contains 11-day data, to re-examine.

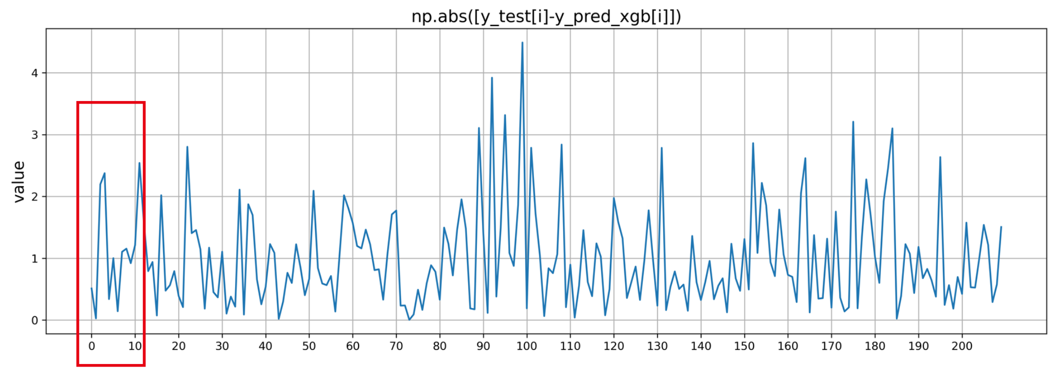
The figures below show the performance of four models and calculating geometric mean predicting 11-day data. It can be seen from figures that the performance of Random Forest Regressor, Linear Regression and Ridge was worse than before (long-term forecasting) but still acceptable. Whereas, the r squared score of XGBoost experienced a sharp decrease, from the highest to the lowest, and it's the only one with a negative score. Although XGBoost didn't perform well, calculating geometric mean still improved the predicted performance compared with the other four models.





5 Conclusion

Comparing the results of using XGBoost model, it seems that this model is better for predicting long-term data. I made a visualisation using the absolute difference between the daily true value and the predicted value when predicting long-term data by using XGBoost. We can find that the errors of the first ten predicted values are relatively large and the errors show a fluctuating trend during the whole period.



What's more, when predicting short-term data, since the total amount of features is small, it is easier to have larger errors than long-term data.

As for geometric mean, I think the reason that no matter how the scores of other models changed, it always improved the prediction is that geometric mean is less affected by extreme values.We can understand this with the help of a formula for calculating the geometric mean:

$$\bar{y} = \sqrt[n]{y_1 \times y_2 \times ... \times y_n}$$

Therefore, we can use this method to smooth the result, thereby reducing overfitting or underfitting to a certain degree.

6 Future Work

Under time constraints, I couldn't try enough models for stock price prediction. In the future, I'd like to implement 'Stacking', 'ensemble learning' and other model mixing methods to see if the result can be improved.

For raw data, I'm going to do some other processing in order to figure out whether the models can perform better for further study.

References

[1] Nabi, R., Ab. M. Saeed, S. & Harron, H. (2020). A Novel Approach for Stock Price Prediction Using Gradient Boosting Machine with Feature Engineering (GBM-wFE). *Kurdistan Journal Of Applied Research, 5(1)*, 28-48. https://doi.org/10.24017/science.2020.1.3

[2] Panwar, B., Dhuriya, G., Johri, P., Singh Yadav, S. & Gaur, N. (2021). Stock Market Prediction Using Linear Regression and SVM. *2021 International Conference On Advance Computing And Innovative Technologies In Engineering (ICACITE)*. https://doi.org/10.1109/icacite51222.2021.9404733

[3] Ariyo, A., Adewumi, A. & Ayo, C. (2014). Stock Price Prediction Using the ARIMA Model. *2014 Uksim-AMSS 16Th International Conference On Computer Modelling And Simulation*. https://doi.org/10.1109/uksim.2014.67

[4] Wang, Y. & Guo, Y. (2020). Forecasting method of stock market volatility in time series data based on mixed model of ARIMA and XGBoost. *China Communications, 17(3)*, 205-221. https://doi.org/10.23919/jcc.2020.03.017

[5] Ghosh, I. & Chaudhuri, T. (2021). FEB-Stacking and FEB-DNN Models for Stock Trend Prediction: A Performance Analysis for Pre and Post Covid-19 Periods. *Decision Making: Applications In Management And Engineering, 4(1)*, 51-86. https://doi.org/10.31181/dmame2104051g

[6] *CSI 300 Index Options | CFFEX*. Cffex.com.cn. (2021). http://www.cffex.com.cn/en_new/CSI300IndexOptions.html.

[7] *Resources - Time Series Project Template - Machine Learning Plus*. Machine Learning Plus. (2021). https://www.machinelearningplus.com/resources/time-series-project-template/.