Capstone Project

Machine Learning Engineer Nanodegree

Stock Price Indicator

Cheng Han Wu

February 1, 2018

I. Definition

Project Overview

In investment and trading, the buy and sell decision have been made by human. But there are lots of factors that influence human and cause loss. So how about machine? there are wealth of information is available in the form of stock prices and company performance, so it is suitable for machine learning.

Here is a paper that using machine learning to predict the stock price, and there are many other papers doing research on this field. So, apply machine learning to predict stock price is feasible.

Investment firms, hedge funds and even individuals have been using financial models to better understand market behavior and models can provide some predicts for trader reference.

We use **TWII(Taiwan Stock Exchange)** and **TSMC(Taiwan Semiconductor Manufacturing Company)** data in this project, TSMC is the target stock we want to predict in this project, the detail about data would be described in **Datasets and Inputs** section below.

Problem Statement

Our problem is that trading decisions made by human is not always reasonable, such as me. So, I want try to use machine learning to train a model which can give the predicted stock price value a day after, and see it can help trader to make better decision or not.

The problem is a regression problem, because the output value which is the price in this project is continuous value. We take features like **Open price**, **Highest price**, **Lowest Price** etc. as input and predict the **Adjusted Close** price value in one day later for example.

In first step, we would fill the null value in dataset, and calculate several technical indicators such as **MACD**, **SMA** etc., add them to training features.

Second, we would choose 8 models which could handle regression problem, the detail would be show in algorithm part below. And compare them with the benchmark model which is the **Linear Regression**.

Third, adjust the features, to see how the performance will be while reduce the features, and decide whether apply the adjustment to final solution or not.

Final, choose the top single model and compare it with ensemble top three models, the better one is our solution model.

Metrics

I use **RMSE** and **R2** as the evaluation metric. Because our problem is a regression problem, we could know how are the performance of the models on this dataset by these metrics.

RMSE (root-mean-square error) is a popular evaluation metric used in regression problem. **RMSE** is the square root of the average of squared errors, the formula is:

$$\sqrt{\frac{\sum_{i=1}^{n}(Predicted - Actual)^2}{N}}$$

And the lower **RMSE** value represent the predicted value has less error, which means the value predicted by the model is close to the actual value.

R2 (the coefficient of determination), It provides a measure of how well future samples are likely to be predicted by the model. Best possible score is 1.0 and it can be negative. And the formula is:

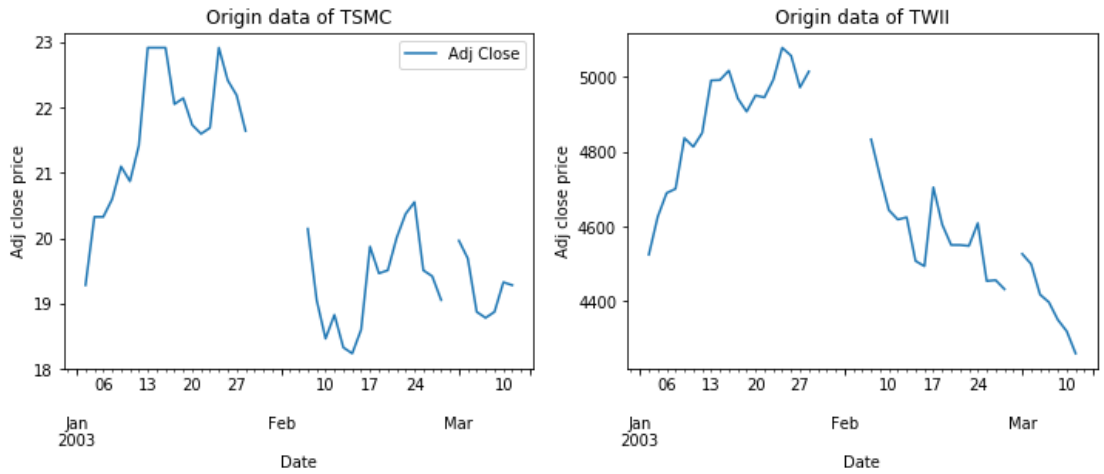
$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=0}^{n_{\text{samples}}-1}(y_i - \hat{y}_i)^2}{\sum_{i=0}^{n_{\text{samples}}-1}(y_i - \bar{y})^2} \quad \text{y is} \quad \bar{y} = \frac{1}{n_{\text{samples}}}\sum_{i=0}^{n_{\text{samples}}-1} y_i$$

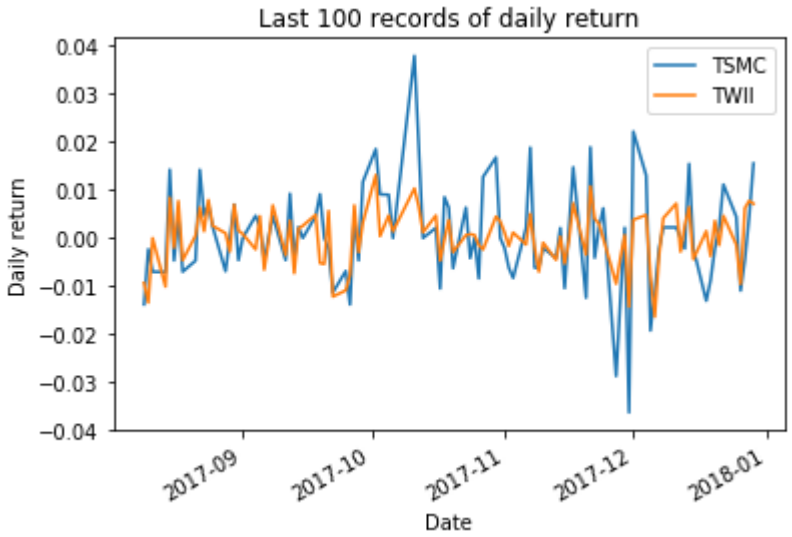
With **RMSE** and **R2** score, when we apply these evaluation metrics to both our solution model and benchmark model, we could compare which one is better.

II. Analysis

Data Exploration

I got data from Yahoo finance, it provides historical data that can be download as csv, I choose **TSMC (Taiwan Semiconductor Manufacturing Company)** which is the weighted stock in Taiwan stock market, so it has high liquidity and it's not easily affected by single major institutional investors.





We can see the general trend of price is same between TSMC and TWII, the space between lines is the day that stock market is close.

The historical data fetch from Yahoo finance has seven columns, **Date, Open, High, Low, Close, Adjusted Close and Volume**, the difference between **Adjusted Close** and **Close** are described here, we would use **Adjusted Close** as our outcome variables which is the value we want to predict. **Open, High, Low, Volume** would be used for train in this project, these features are continuous. **Date** is a kind of categorial data, and we won't directly use it to train, we would use **Date** to sorting the data.

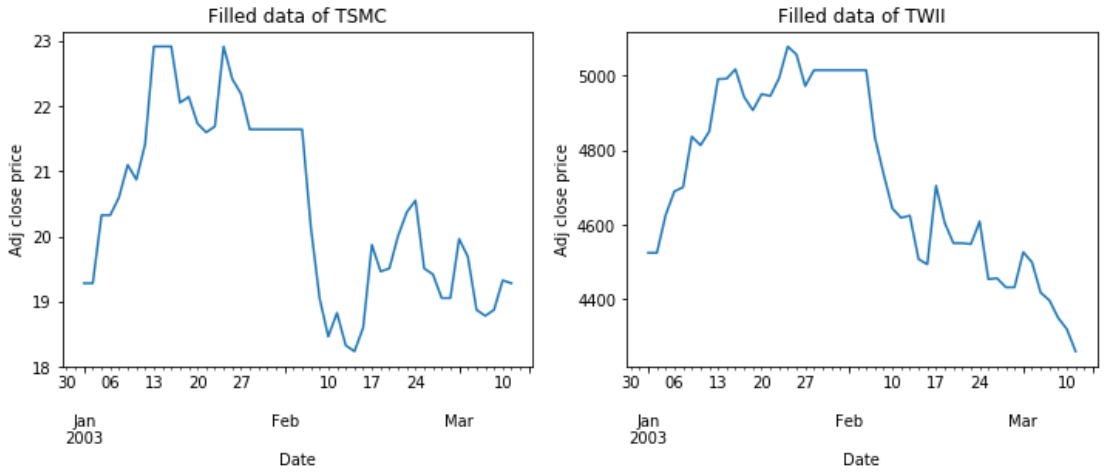
The data date range I choose is from 2003/01/01 to 2017/12/29, the data has 3750 rows in total. Below is the raw data preview.

TSMC data preview

Date	Open	High	Low	Close	Adj Close	Volume
2003-01-01	NaN	NaN	NaN	NaN	NaN	NaN
2003-01-02	33.166302	33.474602	32.703499	32.780399	19.280724	2.992526e+10
2003-01-03	33.937302	34.708401	33.629101	34.554600	20.324272	5.257968e+10
2003-01-06	34.554600	35.248798	34.245602	34.554600	20.324272	4.912915e+10
2003-01-07	35.248798	36.174301	35.017399	35.017399	20.596478	6.169851e+10

As you can see, there are some null values because stock market is close in these days, we would replace the null value with the value of the previous day

or the day after, the reason is that if we just interpolate the null value with mean value, because the mean value is not the actual data of the trade, that would affect the result. And after replacement, the data would look like this.



Except raw data, I add some technical indicators as training features. I choose **Simple Moving Average**, **Bollinger Band**, **MACD** and **RSI**, the details of these indicators would not be described here, you could click the hyper link if you are interested.

Below are the formulas of indicators.

Simple Moving Average

$$\frac{\sum_{i=1}^{n} pi}{n}$$

n is the time series, I choose 15days in the project, so **n** is 15. **p** is the **Adjust Close**.

Bollinger Bands

$$nMA \pm 2 * (n\sigma)$$

n is the same as the time series of **Moving Average**. For example, if we choose 15days, the **Bollinger Bands** are **15MA +- 2 * the Standard Deviation of Adjust Close in 15 days**.

MACD (Moving Average Convergence / Divergence)

$$DIF = EMA(close,12) - EMA(close,26)$$

$$MACD = EMA(DIF,9)$$

MACD has two values, **DIF** and **MACD**. **DIF** is the 12 days **EMA** (exponential moving average) of close price **(Adjusted Close** here) minus 26 days **EMA,** and then **MACD** is the 9 days **EMA** of **DIF.**

RSI (relative strength index)

(When today's price is higher than yesterday)

Up = close(now) - close(previous)

Down = 0

(When today's price is lower than yesterday)

Up = 0

Down = close(previous) - close(now)

$$RS = \frac{EMA(up, n)}{EMA(down, n)}$$

$$RSI = 100 - \frac{100}{1 + RS}$$

RSI is a little bit complex, the **Up** and **Down** have two situations, when the current close price is higher than yesterday, the **Up** would be the close price of today minus the close price of yesterday, the **Down** would be zero. And vice versa. RS is the n days **EMA** of **Up** divided by n days **EMA** of **Down**, n is 14 days in this project. And RSI is just as the formula show above. So, the features would look like the image below.

Date	Open	High	Low	Close	Adj Close	Volume	SMA	Upper_band	Lower_band	DIF	MACD	RSI
2003-02-17	33.782799	33.782799	33.011799	33.782799	19.870312	7.718598e+10	20.200040	23.163993	17.236086	-0.677400	-0.580037	46.308288
2003-02-18	33.782799	33.782799	33.011799	33.088699	19.462057	4.262700e+10	20.054869	22.928577	17.181161	-0.638582	-0.593154	43.285339
2003-02-19	33.860500	33.860500	33.166302	33.166302	19.507700	6.916481e+10	19.912741	22.658578	17.166905	-0.597433	-0.594090	43.725378
2003-02-20	33.166302	34.091801	33.011799	34.014198	20.006414	7.103637e+10	19.803861	22.380544	17.227179	-0.520698	-0.578329	48.411253
2003-02-21	34.708401	35.402500	34.477001	34.631500	20.369501	9.305649e+10	19.719187	22.114416	17.323958	-0.427131	-0.546330	51.557734

Note that, because we calculate the moving average, that means the indicators would have value after the day we specified. Take **MACD** for example, we choose 16, 26 days **DIF**, 9 days **MACD**, so the **DIF** would have value after 26 days, and **MACD** is the EMA of 9 days **DIF**, MACD would have value after 26 plus 9, which is 34 days. In short, we need to drop the first 33 days of data.

Algorithms and Techniques

I suppose to choose different kinds of algorithms, to see if they perform well on this dataset. So, I choose eight different algorithms including **Decision Trees** types, **SVM**, **SGD** and **Ensemble Learner**. Compare their performance, finally choose the one has the best performance or ensemble multiple learners. Following are the algorithms I choose, and their default parameters are in the hyper link, I would list the parameters I modify.

1. **Random Forest Regressor**

   Random Forest are an ensemble learning method, that operate by constructing a multitude of decision trees at training time and outputting the mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set. (reference).

   (random_state=0, random_state is the seed used by the random number generator.)

2. **Linear SVR (Support Vector Regression)**

   The model produced by **Support Vector Classification** depends only on a subset of the training data, because the cost function for building the model does not care about training points that lie beyond the margin. Analogously, the model produced by **Support Vector Regression** depends only on a subset of the training data, because the cost function for building the model ignores any training data close to the model prediction. (reference)

   I have compared SVR, NuSVR and LinearSVR, and choose the one performs the best.

(random_state=0)

3. **SGD (Stochastic Gradient Descent) Regressor**

   SGD stands for **Stochastic Gradient Descent**: the gradient of the loss is estimated each sample at a time and the model is updated along the way with a decreasing strength schedule (aka learning rate).

   And SGD Regressor is a linear model fitted by minimizing a regularized empirical loss with SGD.

   (tol=1e-3, The stopping criterion.

   max_iter=1000, The maximum number of passes over the training data.

   random_state=0)

4. **LassoCV**

   The **Lasso** is a linear model that estimates sparse coefficients. It is useful in some contexts due to its tendency to prefer solutions with fewer parameter values, effectively reducing the number of variables upon which the given solution is dependent. And the best model is selected by cross-validation. (reference)

   (n_alphas=1000, Number of alphas along the regularization path.

   max_iter=3000,

   random_state=0)

5. **RidgeCV**

   Ridge regression addresses some of the problems of Ordinary Least Squares by imposing a penalty on the size of coefficients. (reference)

   (**gcv_mode=auto,** Flag indicating which strategy to use when performing Generalized Cross-Validation.)

6. **AdaBoostRegressor**

   An **AdaBoost** regressor is a meta-estimator that begins by fitting a regressor on the original dataset and then fits additional copies of the regressor on the same dataset but where the weights of instances are

adjusted according to the error of the current prediction. As such, subsequent regressors focus more on difficult cases. (detail) And we use **Linear SVR** as a base estimator of **AdaBoost** here.

(LinearSVR (C=50, epsilon=0, random_state=0), Linear SVR performs the best. So, I choose it to be the base estimator of these ensemble learners.

n_estimators=10, The maximum number of estimators at which boosting is terminated)

7. **BaggingRegressor**

   A **Bagging** regressor is an ensemble meta-estimator that fits base regressors each on random subsets of the original dataset and then aggregate their individual predictions (either by voting or by averaging) to form a final prediction. (detail) Just as **AdaBoost**, we use **Linear SVR** here too.

   (LinearSVR (C=50, epsilon=0, random_state=0)

   n_estimators=10)

8. **XGBRegressor**

   XGBoost is short for “Extreme Gradient Boosting”, where the term “Gradient Boosting” is proposed in the paper *Greedy Function Approximation: A Gradient Boosting Machine*, by Friedman. (reference)

   Here is an introduce about **XGboost**

   (booster='gblinear', which booster to use, can be gbtree, gblinear or dart. gbtree and dart use tree-based model while gblinear uses linear function**)**

All models would take the features below as input,

Open	High	Low	Volume	SMA	Upper_band	Lower_band	DIF	MACD	RSI

And expect to output the **Adjust close** price. We would use **TimeSeriesSplit** function in scikit-learn to split the data into training set and testing set, we split the whole dataset into 10 packs and in each packs, the indices of testing

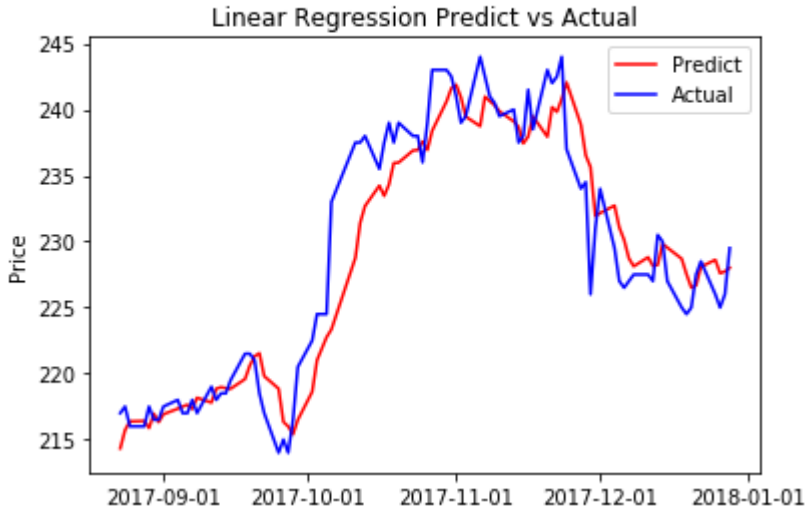
set would be higher than training set. By doing this can prevent **look ahead bias**, which means the model would not use future data to train itself.

I separate the last 90 days data as the validation dataset, to test the models by the data they have never see.

Benchmark

I choose **Linear Regression** as the benchmark, Linear regression is a general model, and usually use as the first model in regression problem to see the performance of the model in the dataset. I use the default parameters and below is the result that the model test by validation set.

```
RMSE:  2.9823229399228497
R2 score:  0.9012439861833962
```



We can see that the model's performance is good actually on validation set, the prices predicted by model are generally fit the trend of the actual price, and next is to see the performance of our solution models.

III. Methodology

Data Preprocessing

We would normalize the data by using **MinMaxScaler** except **RSI**, it could help us transform our data into the range of 0 to 1. The formula is

```
X_std = (X - X.min(axis=0)) / (X.max(axis=0) - X.min(axis=0))

X_scaled = X_std * (max - min) + min
```

RSI value is range from 0~100, and the reason we don't normalize it by **MinMaxScaler** is that the **MinMaxScaler** would normalize the data by its max value and min value, but the actual **RSI** value in our data may not have the value of 0 or 100, it would cause the normalized value is different from the original meaning of **RSI**, so we normalize it to 0~1 by ourselves, divide the **RSI** by 100.

So, the data been processed would look like this.

	Open	High	Low	Volume	SMA	Upper_band	Lower_band	DIF	MACD	RSI
0	0.009425	0.006889	0.006600	0.228361	0.004134	0.011676	0.000315	0.370837	0.341248	0.463083
1	0.009425	0.006889	0.006600	0.126115	0.003479	0.010644	0.000065	0.374641	0.339757	0.432853
2	0.009790	0.007254	0.007334	0.204630	0.002838	0.009459	0.000000	0.378674	0.339651	0.437254
3	0.006526	0.008342	0.006600	0.210167	0.002347	0.008239	0.000274	0.386193	0.341442	0.484113
4	0.013776	0.014505	0.013564	0.275315	0.001965	0.007072	0.000715	0.395362	0.345079	0.515577

Implementation

I define two functions to help training and validating models.

```
def train_clf(model, ts_split):
    clf = model
    for train_index, test_index in ts_split.split(feature_minmax_transform):
        X_train, X_test = feature_minmax_transform[:len(train_index)], feature_minmax_transform[len(train_index): (len(train_index)+len(test_index))]
        y_train, y_test = target_adj_close[:len(train_index)], target_adj_close[len(train_index): (len(train_index)+len(test_index))]
        clf.fit(X_train, y_train)
    return clf

def validate_result(model, model_name):
    predicted = model.predict(validation_X)
    RSME_score = np.sqrt(mean_squared_error(validation_y, predicted))
    print('RMSE: ', RSME_score)

    R2_score = r2_score(validation_y, predicted)
    print('R2 score: ', R2_score)

    plt.plot(predicted,'r', label='Predict')
    plt.plot(validation_y,'b', label='Actual')
    plt.xlabel('Index')
    plt.ylabel('Price')
    plt.title(model_name + ' Predict vs Actual')
    plt.legend(loc='upper right')
    plt.show()
```

train_clf take a model and a time-split series object as parameter, it would split the data by the time-split series, and train the model.

validate_result take a model and the model's name as input, it would use validation set to see the performance of model, and output the RMSE error and R2 score, also plot the result.

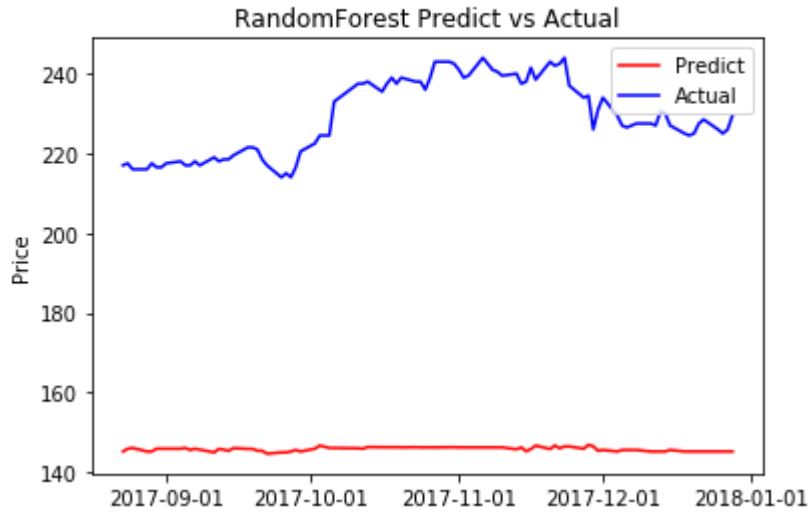
We train our solution models, and use **GridSearchCV** to adjust the parameters. Below are the performances of models, their parameters and the ranking of **RMSE** error.

Random Forest Regressor

The result of default parameter is not ideal, it didn't learn how to predict price. So, let's try **GridSearchCV** to adjust the parameters, see whether could we improve the model. the parameters are listed here.

```
random_forest_parameters = {
    'n_estimators':[10, 50, 100, 300],
    'max_features':['auto','sqrt','log2'],
    'max_depth':[None, 3, 5, 7],
}
```

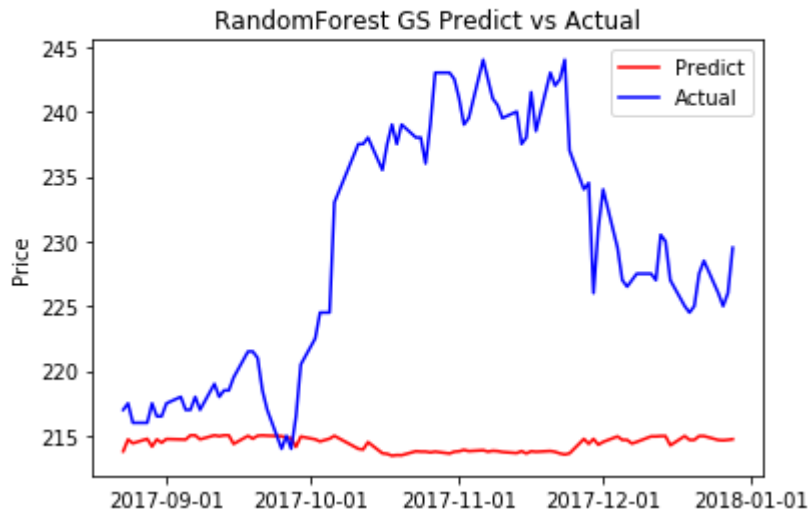
```
RMSE:  83.81106871226116
R2 score:  -76.99324152589374
```



The best parameters set find out by **GridSearchCV** is:

{'max_depth': 7, 'max_features': 'auto', 'n_estimators': 100,'random_state': 0}

```
RMSE:  17.72029911425973
R2 score:  -2.4865533125528687
```



But the result is still not good, maybe **RandomForest** is not suitable for this problem or dataset, let's try other algorithms first.

Linear SVR (Support Vector Regression)

After comparing three kinds of SVR, we still try GridSearchCV to find out the best parameters set.

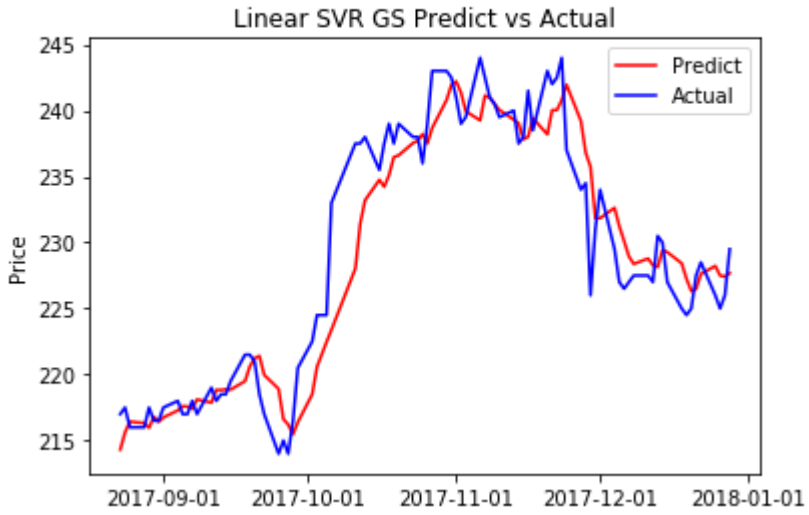
	NuSVR	**SVR**	**LinearSVR**
RMSE	4.6300	4.2228	4.0877
R2	0.7619	0.8019	0.8144

```
linear_svr_parameters = {
    'C':[0.5, 1.0, 10.0, 50.0],
    'epsilon':[0, 0.1, 0.5, 0.7, 0.9],
}
```

The best parameters set is:

{'C': 50.0, 'epsilon': 0.1, 'random_state': 0}

```
RMSE:  2.9618035028517533
R2 score:  0.9025982637823058
```



This time, the performance of the model using adjusted parameters have a significant progress.

SGD (Stochastic Gradient Descent) Regressor

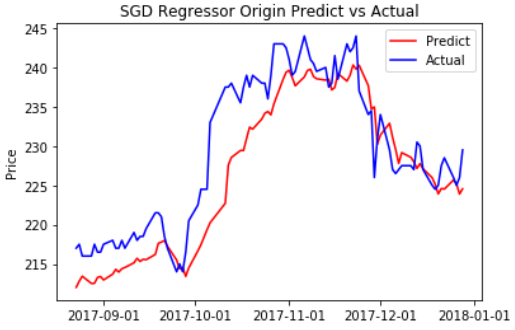
This is the parameters of SGD

```
sgdr_parameters = {
    'loss':['squared_loss', 'huber', 'epsilon_insensitive', 'squared_epsilon_insensitive'],
    'penalty':['none', 'l2', 'l1', 'elasticnet'],
    'alpha':[0.0001, 0.001, 0.01, 0.5],
    'learning_rate':['constant','optimal','invscaling'],
    'average':[0, 1, 5, 10]
}
```

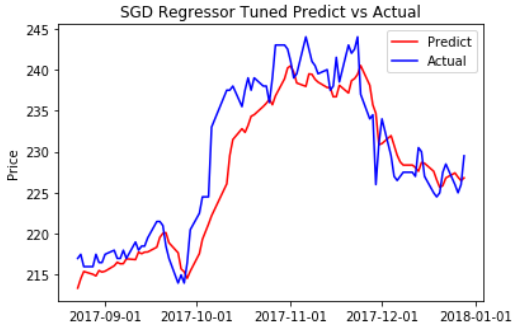
Best parameters set:

{'alpha': 0.0001, 'average': 0, 'learning_rate': 'optimal', 'loss': 'epsilon_insensitive', 'penalty': 'l1', tol': 1e-3, ' max_iter ': 1000 'random_state': 0}

```
RMSE:  4.448599411568858
R2 score:  0.7802642027961754
```




```
RMSE:  3.463938498353611
R2 score:  0.8667722913207112
```



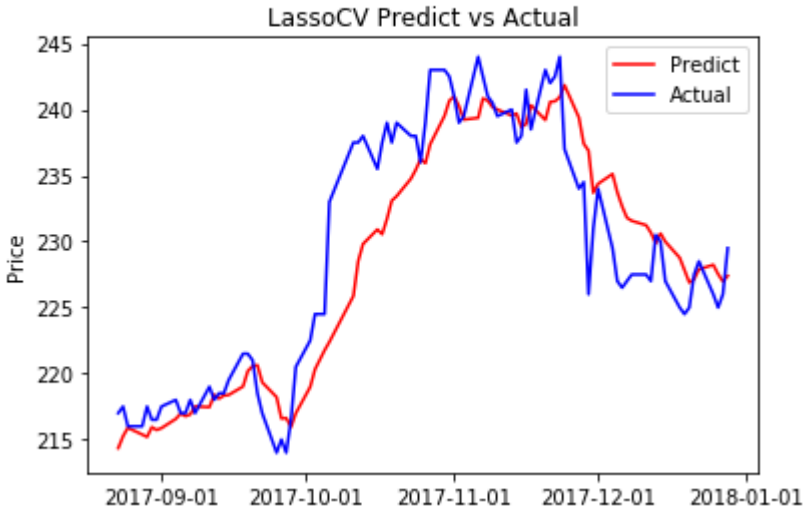
The left figure is the original one, and the right figure is the tuned one. Still have great advancement.

LassoCV

By default, it performs Cross-Validation, so I just tune some parameters. After the trying, I found out that n_alpha sets to 1000 perform better, and set the max_iter 3000, because if the max_iter value is low, the model would reach the limit before convergence.

{'n_alpha': 1000, 'max_iter': 3000, 'random_state'=0}

```
RMSE:  3.765758205688586
R2 score:  0.842544050284762
```

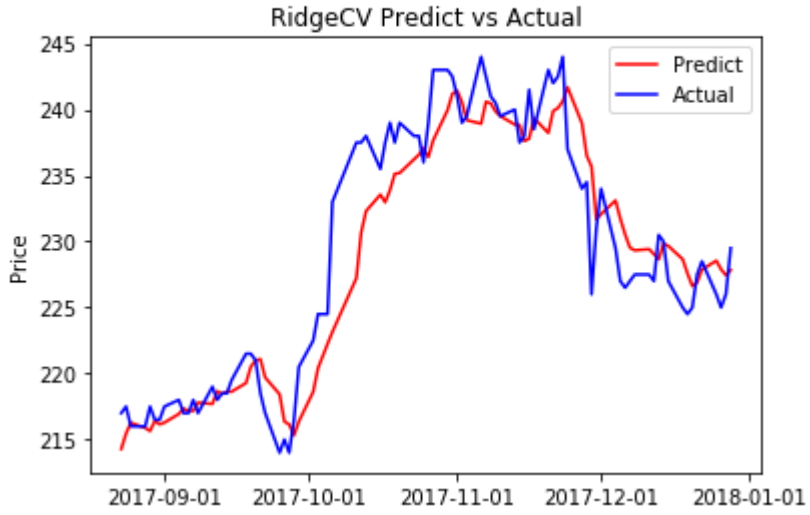


RidgeCV

This model is a little bit like **LassoCV,** it performs Generalized Cross-Validation. I use all default parameters.

{'gcv_mode': 'auto'}

```
RMSE:  3.175064870106251
R2 score:  0.8880666712677514
```



The performance is good.

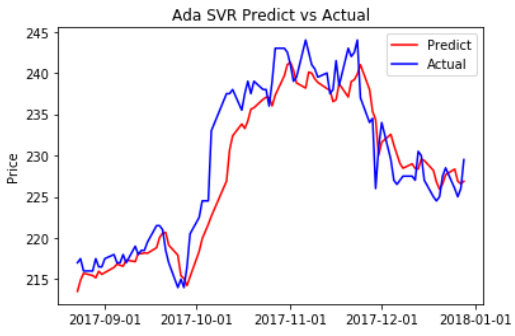
AdaBoostRegressor and **BaggingRegressor**

These two ensemble methods use the same parameters, so I put them together. I use the **Linear SVR** which has the best score now as the base estimator. I have tried n_estimators in [5, 10, 50, 100], and that 10 has the best score.

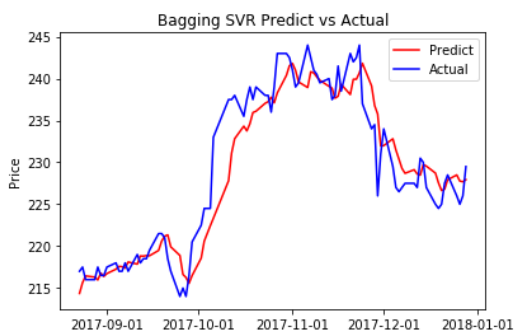
{'base_estimator': LinearSVR ('C'=50, 'epsilon'=0, 'random_state'=0)

'n_estimators':10,}

```
RMSE:  3.2061092216759977
R2 score:  0.8858671032319537
```




```
RMSE:  3.0502953223606393
R2 score:  0.8966910396499718
```



These two ensemble learners have good score.

XGBRegressor

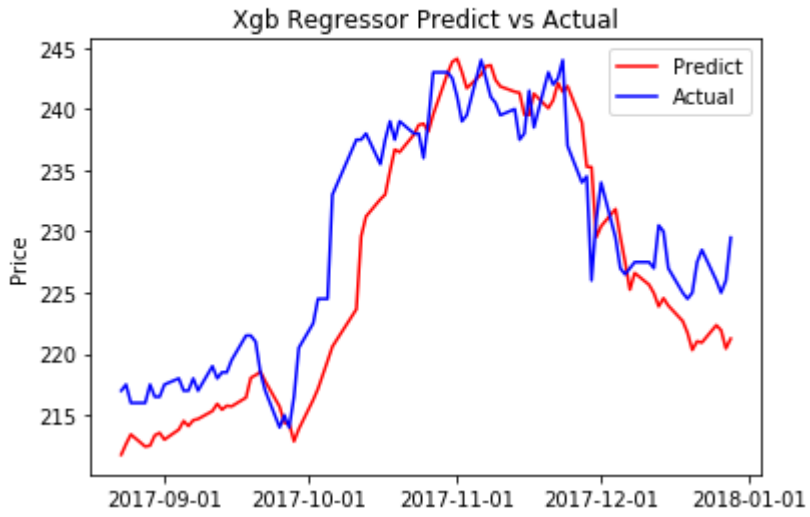
There are many parameters in XGBoost can be adjust, so I use GridSearch to find.

```
xgb_params = {
    'colsample_bytree':[0.3,0.5,0.7,1],
    'colsample_bylevel':[0.3,0.5,0.7,1],
    'gamma':[0,0.3,0.5,1],
    'max_depth': [3,6,9],
    'n_estimators':[50,100,150,200],
    'reg_lambda': [0.1,0.3,0.5],
    'learning_rate':[0.1,0.3,0.5],
    'reg_alpha':[0,0.1,0.5],
}
```

After taking a long time, the best parameters set is:

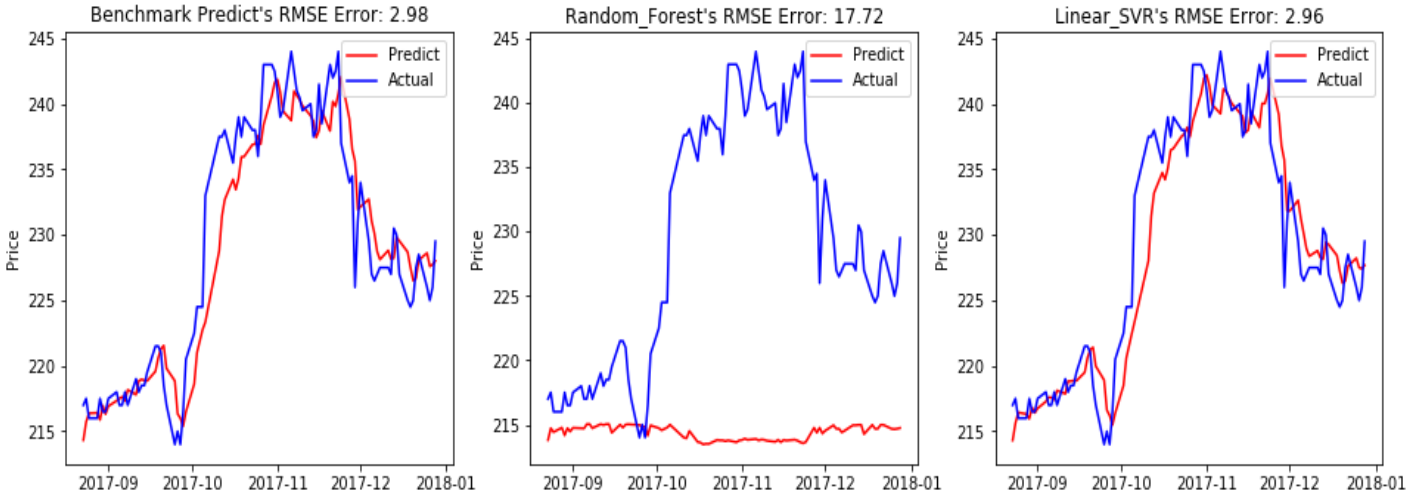
{'booster': 'gblinear', 'colsample_bylevel': 0.3, 'colsample_bytree': 0.3, 'gamma':0,'learning_rate': 0.5, 'max_depth': 3, 'n_estimators': 200, 'reg_alpha': 0, 'reg_lambda': 0.1}

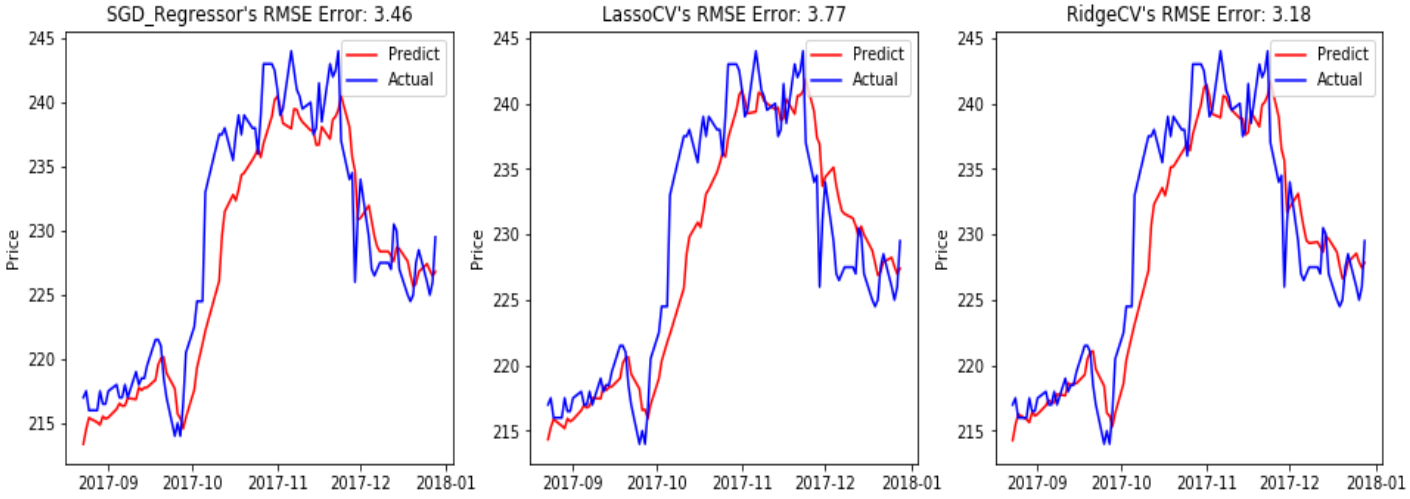
```
RMSE:  4.2308379336035555
R2 score:  0.801250066186511
```

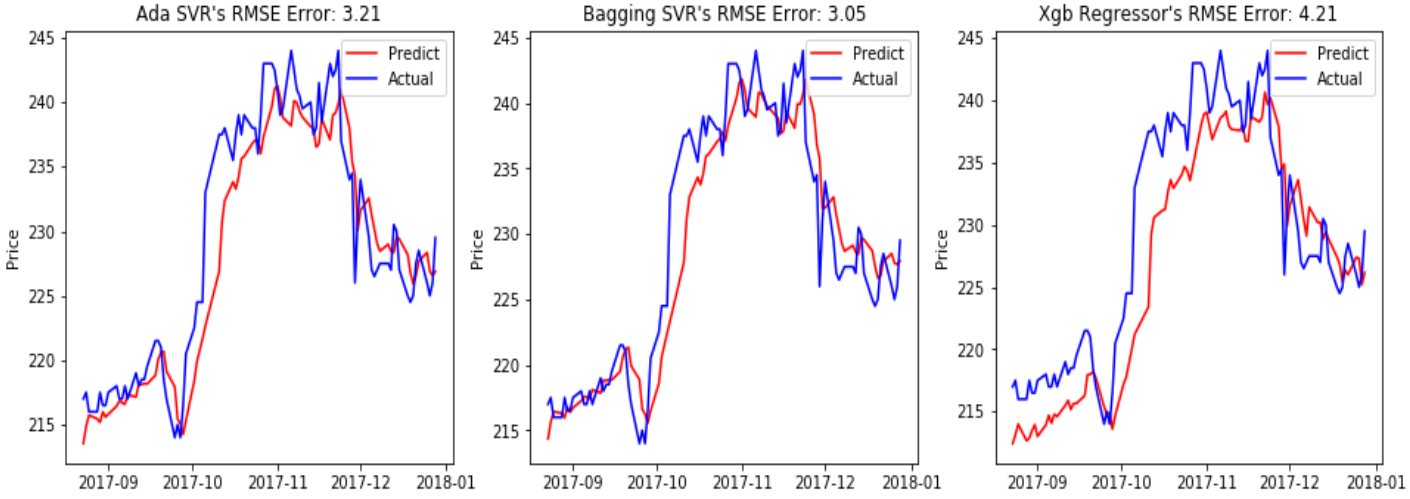


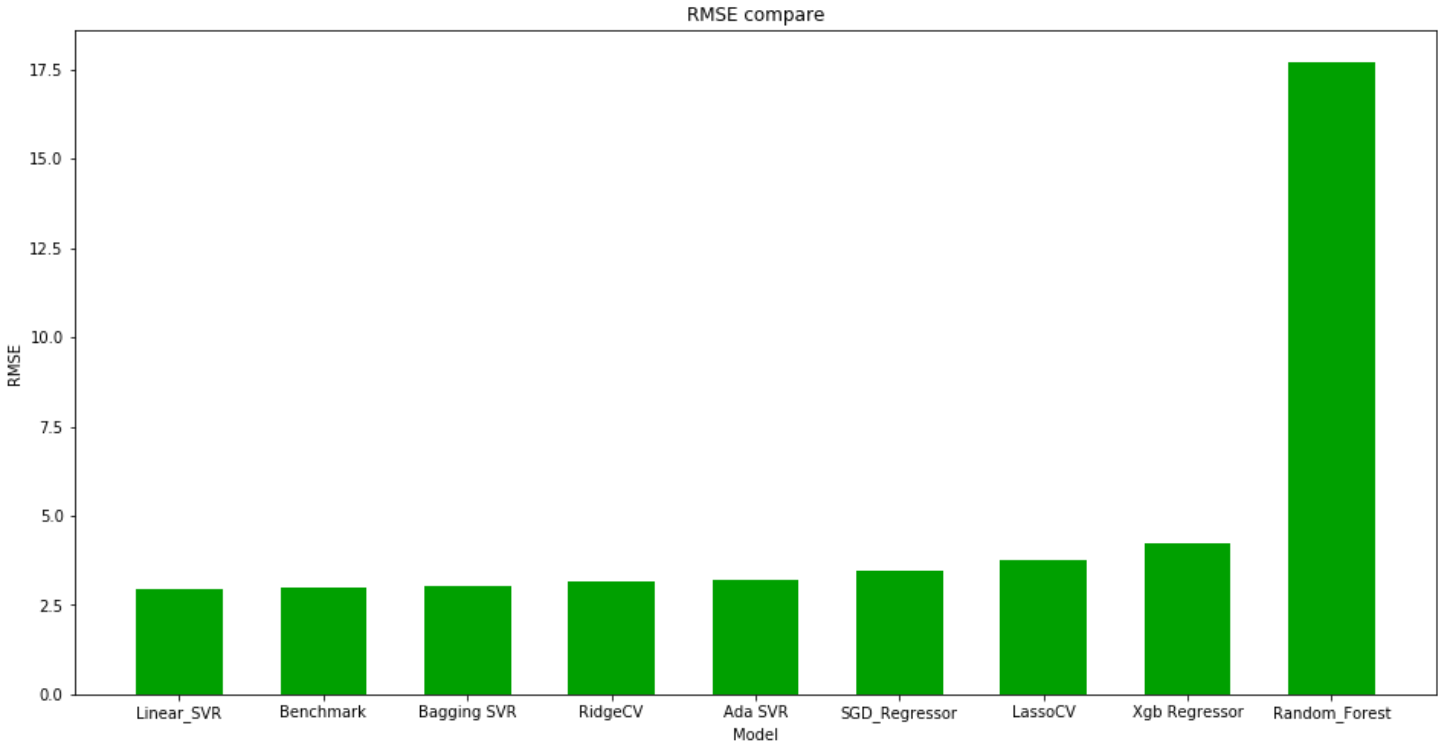
It's not bad, but it is worse than some models above.

Below is the summarize of these models, and a ranking by RMSE error.


Benchmark Predict's RMSE Error: 2.98
Random_Forest's RMSE Error: 17.72
Linear_SVR's RMSE Error: 2.96
Predict
Actual
Price
245
240
235
230
225
220
215
2017-09
2017-10
2017-11
2017-12
2018-01


SGD_Regressor's RMSE Error: 3.46
LassoCV's RMSE Error: 3.77
RidgeCV's RMSE Error: 3.18
Predict
Actual
Price
245
240
235
230
225
220
215
2017-09
2017-10
2017-11
2017-12
2018-01


Ada SVR's RMSE Error: 3.21
Bagging SVR's RMSE Error: 3.05
Xgb Regressor's RMSE Error: 4.21
Predict
Actual
Price
245
240
235
230
225
220
215
2017-09
2017-10
2017-11
2017-12
2018-01



We can see that the RSME errors of models are close except **Random Forest**, and next we see if we reduce the features, would the performance of models be better or worse.

Refinement

I choose **SelectFromModel** as our feature selection tool, it can selecting features based on importance weights, so we need to pass the model which has the feature importance attribute. I choose **RidgeCV** here, because in the models which has feature importance or coef attribute, it has the best performance. And use the default **threshold** (The threshold value to use for feature selection. Features whose importance is greater or equal are kept while the others are discarded).

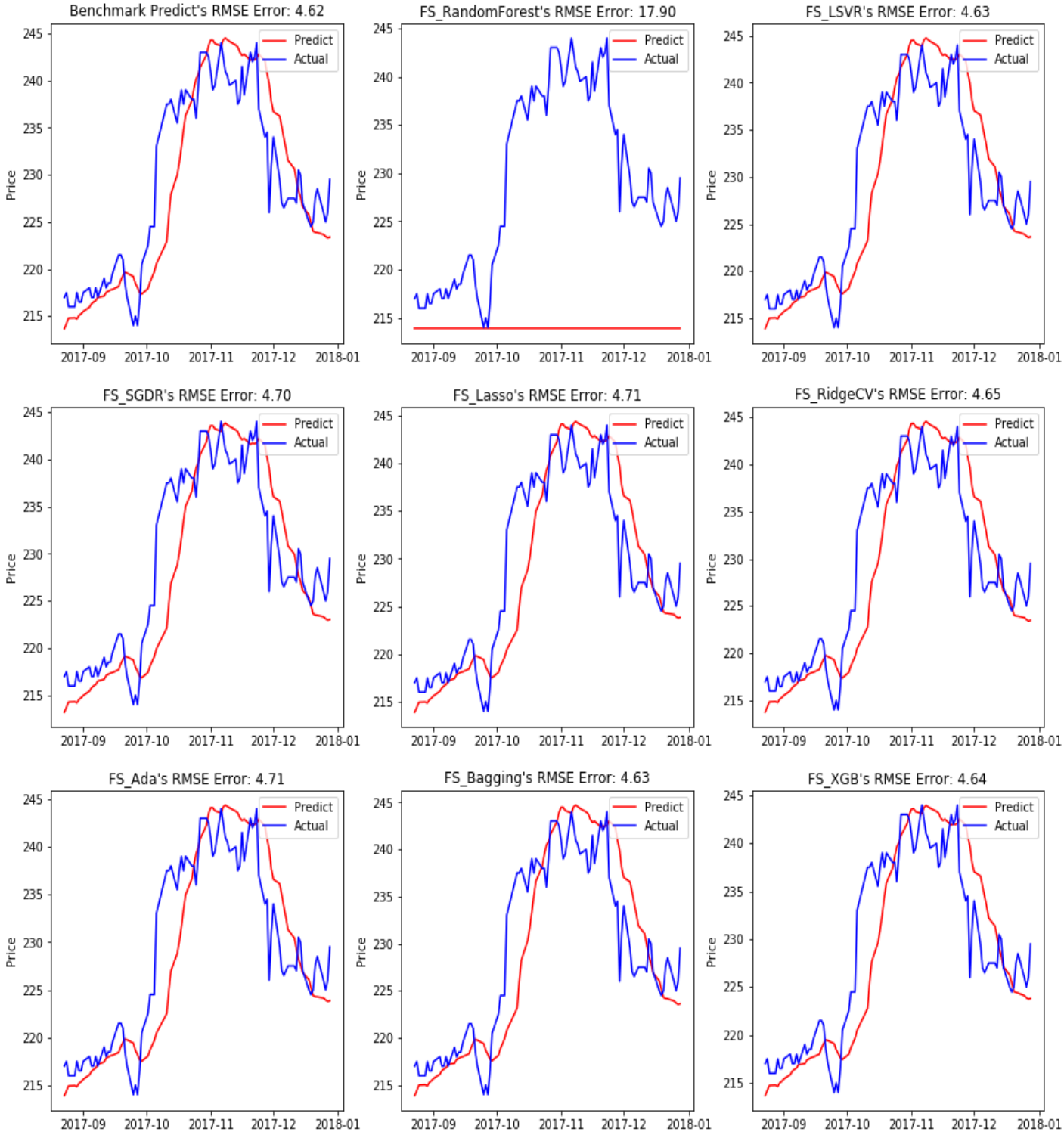
Date	Open	High	Low	Volume	SMA	Upper_band	Lower_band	DIF	MACD	RSI
2003-02-17	0.009425	0.006889	0.006600	0.228361	0.004134	0.011676	0.000315	0.370837	0.341248	0.463083
2003-02-18	0.009425	0.006889	0.006600	0.126115	0.003479	0.010644	0.000065	0.374641	0.339757	0.432853
2003-02-19	0.009790	0.007254	0.007334	0.204630	0.002838	0.009459	0.000000	0.378674	0.339651	0.437254
2003-02-20	0.006526	0.008342	0.006600	0.210167	0.002347	0.008239	0.000274	0.386193	0.341442	0.484113
2003-02-21	0.013776	0.014505	0.013564	0.275315	0.001965	0.007072	0.000715	0.395362	0.345079	0.515577

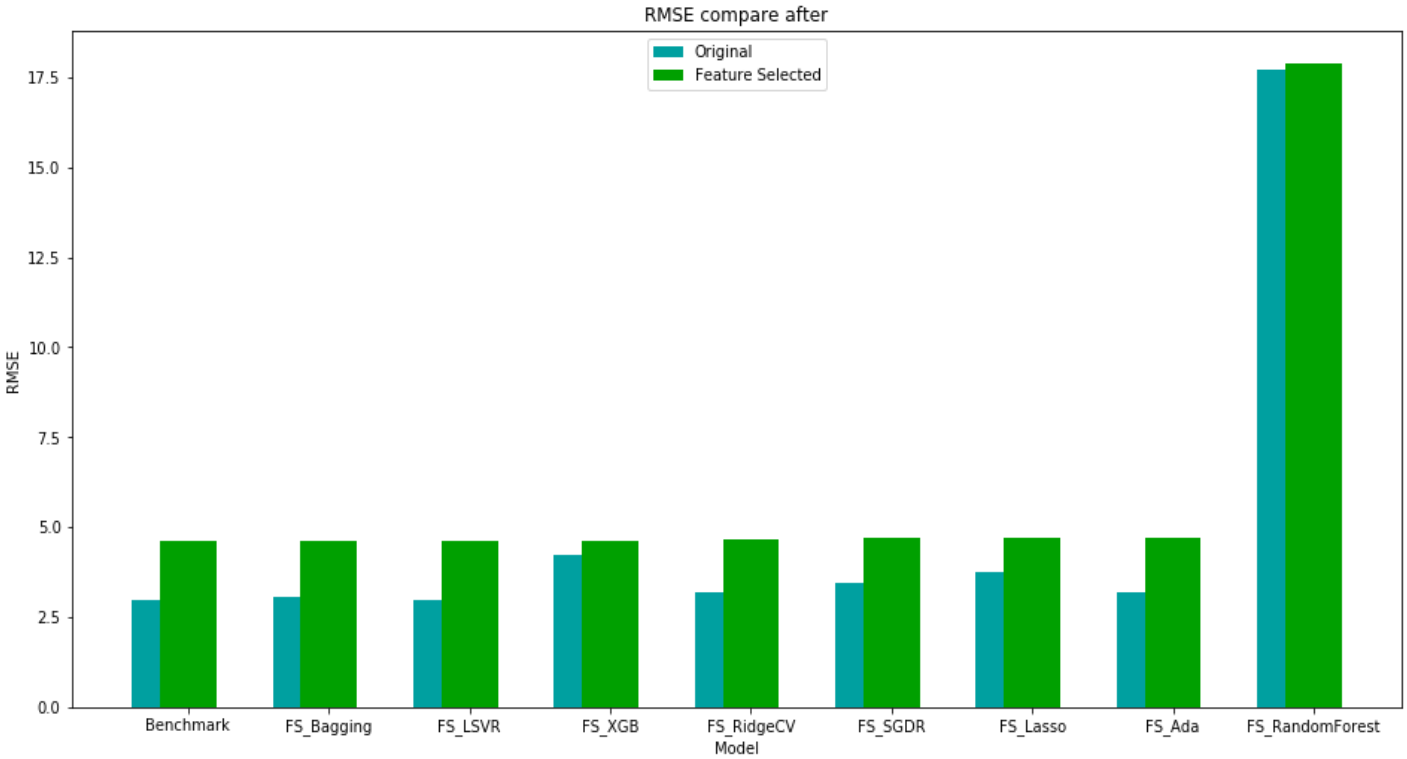
```
('Open', False) ('High', False) ('Low', False) ('Volume', False) ('SMA', True) ('Upper_band', True) ('Lower_band', True) ('DI
F', True) ('MACD', False) ('RSI', False)
```

The **True** and **False** value show that the feature has been selected or not. So, we create a new data set by the selection result, and it look like the image below.

Date	SMA	Upper_band	Lower_band	DIF
2003-02-17	0.004134	0.011676	0.000315	0.370837
2003-02-18	0.003479	0.010644	0.000065	0.374641
2003-02-19	0.002838	0.009459	0.000000	0.378674
2003-02-20	0.002347	0.008239	0.000274	0.386193
2003-02-21	0.001965	0.007072	0.000715	0.395362

After selected features, we train all models and validate it by the selected features dataset to see the changes.


Benchmark Predict's RMSE Error: 4.62
FS_RandomForest's RMSE Error: 17.90
FS_LSVR's RMSE Error: 4.63
FS_SGDR's RMSE Error: 4.70
FS_Lasso's RMSE Error: 4.71
FS_RidgeCV's RMSE Error: 4.65
FS_Ada's RMSE Error: 4.71
FS_Bagging's RMSE Error: 4.63
FS_XGB's RMSE Error: 4.64
Predict
Actual
Price
245
240
235
230
225
220
215
2017-09
2017-10
2017-11
2017-12
2018-01



As the result shows, the original dataset performs better than the feature selected dataset. If we drop some features, the model would also lose some important information. This is the running time I test on the ensemble model on **LinearSVR**, **Bagging Regressor**, **RidgeCV**

Original one
Running Time: 0.0015022754669189453

FeatureSelected
Running Time: 0.0010027885437011719

We can see that the running time with only a slight difference, so that I decided to use the original feature set.

IV. Results

Model Evaluation and Validation

Final, as the mention before, we pick up the top three models in the ranking, which is the **Bagging Regressor, LinearSVR** and **AdaBoost**. And compare this ensemble model with single **LinearSVR.**

We create a class that ensemble these three models, it would take input, and average the result predicted by model.

```
# Choose the top three model to ensemble them
ensemble_solution_models = [lsvr_grid_search, bagging_clf, ridge_clf]
class EnsembleSolution:
    models = []
    def __init__(self, models):
        self.models = models
    def fit(self, X, y):
        for i in self.models:
            i.fit(X, y)
    def predict(self, X):
        start = time.time()
        result = 0
        for i in self.models:
            result = result + i.predict(X)

        result = result / len(self.models)
        end = time.time()
        print('Running Time: {}'.format(end - start))
        return result
```

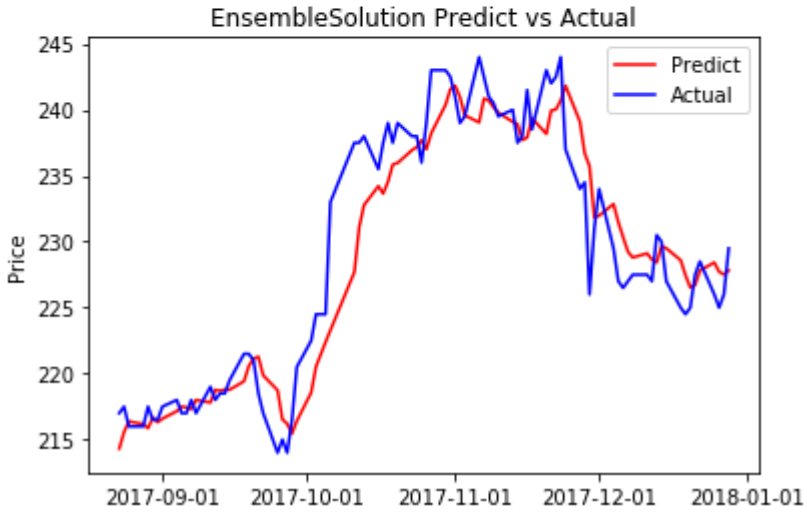
And we validate this ensemble solution.

```
Compare on original features

Benchmark Model
RMSE:  2.9823229399228497
R2 score:  0.9012439861833962

LinearSVR Model
RMSE:  2.9618035028517533
R2 score:  0.9025982637823058

Ensemble Solution Model
Running Time: 0.0010018348693847656
RMSE:  3.0543951470010837
R2 score:  0.8964131431075624
```



Both the performances of **Benchmark** and **LinearSVR** is a little bit better than the **Ensemble** model, but there is another step to do, to check whether the **Ensemble** model would be more robust or not. We check this through training them on random timeseries split and also test on random split, and take average to the **RMSE** and **R2.**

First, we train **Benchmark**, **LinearSVR** and the **Ensemble** model on different data split. By the **train_clf_multipletimes** function. If the image is not clear, could see it in notebook, this function would train the pass in model several times (5 times I use), and use different parameters on TimeSeriesSplit in each time, average the **R2** and **RMSE**.

```python
def train_clf_multipletimes(model, times):
    total_rmse = 0
    total_r2 = 0
    for i in range(times):
        clf = model
        for train_index, test_index in TimeSeriesSplit(n_splits=i+2).split(feature_minmax_transform):
            X_train, X_test = feature_minmax_transform[:len(train_index)], feature_minmax_transform[len(train_index): (len(train_index)+len(test_index))]
            y_train, y_test = target_adj_close[:len(train_index)].values.ravel(), target_adj_close[len(train_index): (len(train_index)+len(test_index))].values.ravel()
            clf.fit(X_train, y_train)
        predicted = clf.predict(validation_X)
        rmse, r2 = print_result(validation_y, predicted, [0,len(validation_y)])
        total_rmse += rmse
        total_r2 += r2
    return total_rmse / times, total_r2 / times
```

```
Benchmark:
RMSE: 3.26496509019226 // R2: 0.8811605433575804
Ensemble
RMSE: 3.4153404699098973 // R2: 0.8697569023887907
LSVR
RMSE: 3.3258300973356527 // R2: 0.8767469184643775
```

Above is the result of training multiple times on different cross validation, the benchmark performs the best, and the LSVR is a little bit better than ensemble learner.

Final, we again try these three models on different split of validation set, after this step, we would choose the **LSVR** or **Ensemble** model as our solution model. This function is a little bit like the function above, we compute the average of **RSME** and **R2** in total.

```python
def cross_validate(model, ts_split):
    clf = model
    total_rmse = 0
    total_r2 = 0
    count = 0
    for train_index, test_index in ts_split.split(validation_X):
        X_test1, X_test2 = validation_X[:len(train_index)], validation_X[len(train_index): (len(train_index)+len(test_index))]
        y_test1, y_test2 = validation_y[:len(train_index)].values.ravel(), validation_y[len(train_index): (len(train_index)+len(test_index))].values.ravel()
        predicted_test1 = clf.predict(X_test1)
        temp1_RMSE, temp1_R2 = print_result(y_test1, predicted_test1, train_index)

        predicted_test2 = clf.predict(X_test2)
        temp2_RMSE, temp2_R2 = print_result(y_test2, predicted_test2, test_index)

        total_rmse += temp1_RMSE + temp2_RMSE
        total_r2 += temp1_R2 + temp2_R2
        count += 2
    return total_rmse / count, total_r2 / count

def print_result(actual, predict, index):
    RMSE_score = np.sqrt(mean_squared_error(actual, predict))
    print('From {} to {}'.format(index[0],index[-1]))
    print('RMSE: ', RMSE_score)
    R2_score = r2_score(actual, predict)
    print('R2 score: ', R2_score)
    print('--------------------')
    return RMSE_score, R2_score
```

(I also print out the detail RMSE and R2 in each split, it is too long, so I just put the average here.)

```
Benchmark RMSE: 2.6744083445881563 // Benchmark R2: -0.5749556227753002
Ensemble RMSE: 2.867739863316714 // Ensemble R2: -0.903554789628217
LSVR RMSE: 2.6487740234952946 // LSVR R2: -0.4150489257089208
```

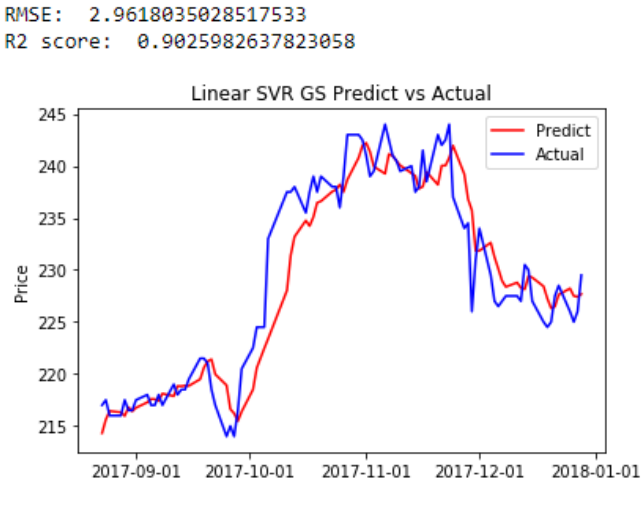
We could see that LSVR's performance is still better than the ensemble model, the reason maybe is that we include different kinds of algorithms together, such as **RidgeCV** and **Bagging SVR**, and their performance are not stable each time, so when we average the score, it would lower than the single good model like **LinearSVR.**

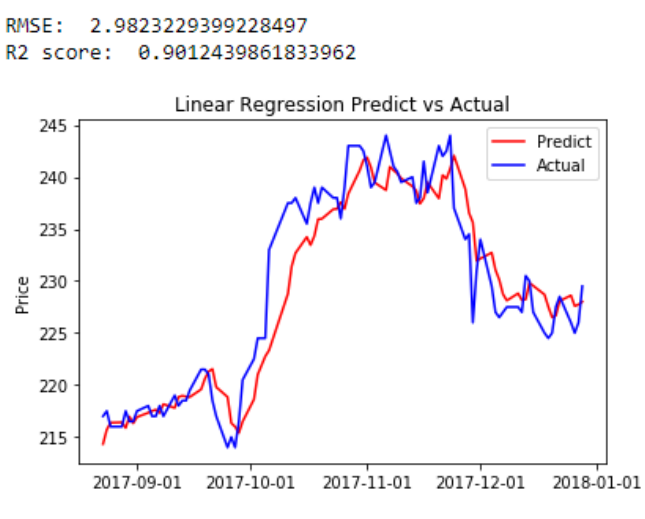
So, by the result above, I choose **LinearSVR** as our final solution model

V. Conclusion

Reflection

We finally make a model that could take data input, and could predict the price 1 day after, although it's not very precision. There is an interesting point is that the value predicted by all models in this project is always slower than the actual value, and the models perform worse in the situation that the price rise or down extremely.





And the difficult point is that, it is impossible to get a model that can 99% predict the price without any error, there are too many factors can affect the market, such as disasters, political factors etc. So, we cannot hope there is a perfect model, but this solution model indeed reached the goal in this project, which is help trader to make better decision. The general trend of predicted price is in line with the actual data, so the trader could have an indicator to reference, and makes trading decision by himself.

Improvement

There are tons of indicators that I have not use in this project, maybe there are some indicators are more suitable to be used in Machine Learning.

And also, there are more algorithms I have not use, maybe the neural network approach is better than the approach of mine. So, there are plenty of ways for improvement.