ORIGINAL RESEARCH PAPER

Machine Learning

INVESTIGATING THE EFFECTIVENESS OF DIFFERENT MACHINE LEARNING ALGORITHMS TO PREDICT THE PRICES OF STOCKS

KEY WORDS: Machine Learning Algorithms, Hyperparameters, Stocks, Features

Ruchit Karnik Indian School Wadi Kabir [Cambridge] (Student)

ABSTRACT

The fluctuating nature of stock prices creates a sense of uncertainty and uneasiness for investors. It is extremely difficult to predict stock prices as price movements are a result of the mass psychology of buyers and sellers. To address this issue, this study aims to identify the best machine learning model to accurately predict stock prices of 5 fortune 500 companies which include Amazon, Meta, Tesla, Apple and Nvidia. The machine learning models used are Linear regression, Decision trees, Random forests, XGboost and K-Nearest Neighbours. Ridge regression was also used in the beginning, but was discontinued due to very identical results to linear regression in order to save computing power and time. All models performed perfectly well and the performance improved significantly using hyperparameters through the grid search feature. Comparing all models, the Random forests model performed the best among all. The performance of the models can be improved by using a dataset with more features and increasing the number of hyperparameters for all models. Moreover, the random forest model can be extended into an application that provides insights about stock prices for investors

INTRODUCTION

Since the beginning of the stock market, predicting stock prices has been of great interest to economists, investors and traders. Early methods to predict stock prices include fundamental analysis and technical analysis amongst others. Fundamental analysis consists of analysing a company's financial statements, ratios and economic indicators, while technical analysis comprises of examining chart patterns. Although these methods have seen some success over the years, they are known to be extremely unreliable and misleading due to oversimplifying the factors that determine stock prices. (Remesh, n.d.) To overcome such limitations of traditional prediction methods, algorithmic trading was conceived. This involves the use of computer programs to analyse historic values of features that influence stock prices to foresee them. In fact, this research involves the use of basic algorithmic trading algorithms to compare the effectiveness of different algorithms.

DATASETS

The datasets were obtained for the following companies: Amazon, Apple, Tesla, Meta and Nvidia. Each dataset includes information about the stock for each day for the last 5 years. The information included for each day is the date, the open price, the high price, the low price, the close price, the adjusted close price, and the volume. All datasets were obtained from Yahoo finance due to its credibility and the datasets were all stored as CSV files. In order to make the predictions more accurate some common features were derived from the information available. The raw Amazon and Apple datasets downloaded from Yahoo finance are shown as an example below.

	A	B	C	D	E	F	G
1	Date	Open	High	Low	Close	Adj Close	Volume
2	13/06/2019	93.336	94.1545	93.111	93.515	93.515	55916000
3	14/06/2019	93.2	93.8	92.95	93.4835	93.4835	57024000
4	17/06/2019	93.825	94.7845	93.7725	94.3015	94.3015	52686000
5	18/06/2019	95.0675	96.0835	94.9895	95.0685	95.0685	77914000
6	19/06/2019	95.392	95.979	94.6235	95.4395	95.4395	57906000
7	20/06/2019	96.6665	96.76	95.29	95.9095	95.9095	64344000
8	21/06/2019	95.805	96.2975	95.379	95.565	95.565	78672000
9	24/06/2019	95.633	95.843	95.065	95.695	95.695	45660000
10	25/06/2019	95.592	95.8195	93.621	93.9135	93.9135	60246000
11	26/06/2019	94.624	95.19	94.366	94.8915	94.8915	48838000
12	27/06/2019	95.1	95.562	94.902	95.214	95.214	42834000
13	28/06/2019	95.455	95.647	94.2	94.6815	94.6815	60748000
14	01/07/2019	96.149	96.491	95.733	96.1095	96.1095	63842000
15	02/07/2019	95.969	96.7395	95.3315	96.7155	96.7155	52918000

Figure 1: Screenshot of Raw Amazon CSV File

	A	B	C	D	E	F	G
1	Date	Open	High	Low	Close	Adj Close	Volume
2	17/06/2019	48.225	48.74	48.0425	48.4725	46.84412	58676400
3	18/06/2019	49.0125	50.0725	48.8025	49.6125	47.94582	106204000
4	19/06/2019	49.92	49.97	49.3275	49.4675	47.8057	84496800
5	20/06/2019	50.0925	50.1525	49.5075	49.865	48.18985	86056000
6	21/06/2019	49.7	50.2125	49.5375	49.695	48.02556	191202400
7	24/06/2019	49.635	50.04	49.5425	49.645	47.97724	72881600
8	25/06/2019	49.6075	49.815	48.8225	48.8925	47.25001	84281200
9	26/06/2019	49.4425	50.2475	49.3375	49.95	48.27199	104270000
10	27/06/2019	50.0725	50.3925	49.8925	49.935	48.2575	83598800
11	28/06/2019	49.67	49.875	49.2625	49.48	47.81777	124442400
12	01/07/2019	50.7925	51.1225	50.1625	50.3875	48.69479	109012000
13	02/07/2019	50.3525	50.7825	50.34	50.6825	48.97989	67740800
14	03/07/2019	50.82	51.11	50.6725	51.1025	49.38577	45448000
15	05/07/2019	50.8375	51.27	50.725	51.0575	49.34229	69062000

Figure 2: Screenshot of raw Apple CSV file

METHODOLOGY

Features

The datasets were all downloaded as csv files and added to a common PyCharm project. The following features were used to train the models: price change, difference between high and low price, Relative Strength Index (RSI) and volume. Only volume was used from the raw data, rest of the features were derived from the data. The price change was derived by subtracting the closing price of one day from the previous day. The difference between the high and low price was derived by subtracting the high price from the low price. Deriving the RSI had a complex procedure (howtoexcel, n.d.). The RSI was calculated over a 14-day period. A column was made for each the gains and the losses in price using the price change column. If the there was a price gain, it was included in the gains column and 0 was included in the losses column, while if there was a price loss, the absolute value of the loss was included in the losses column and 0 was included in the gains. The below figure shows an example of this for Amazon stock.

Date	price change	Gains	Losses
14/06/2019	-0.031501	0	0.031501
17/06/2019	0.818	0.818	0
18/06/2019	0.766999	0.766999	0
19/06/2019	0.371002	0.371002	0
20/06/2019	0.470001	0.470001	0
21/06/2019	-0.344498	0	0.344498
24/06/2019	0.129998	0.129998	0
25/06/2019	-1.781502	0	1.781502

26/06/2019	0.978004	0.978004	0
27/06/2019	0.322495	0.322495	0
28/06/2019	-0.532494	0	0.532494

Figure 3: Example Of The Gains And Losses In The Amazon Stock

Furthermore, average gain and average loss columns were made using data from the gains and losses columns. The average gain column includes the average of 14 days of gains, for example the 14th day includes the average of the gains from the 1st day to the 14th day and the 15th day includes the average of the gains from the 2nd day to the 14th day. The same logic applies to the average loss column. An example of this is shown below from the Amazon stock file

Date	price change	Gains	Losses	Avg gain	Avg loss
04/10/2019	0.761497	0.761497	0	0.35067	0.537643
07/10/2019	-0.349495	0	0.349495	0.325623	0.562607
08/10/2019	-1.357506	0	1.357506	0.302364	0.641393
09/10/2019	0.824006	0.824006	0	0.339624	0.641393
10/10/2019	-0.086503	0	0.086503	0.315365	0.549929
11/10/2019	0.583001	0.583001	0	0.334482	0.518286
14/10/2019	0.225502	0.225502	0	0.326698	0.36225
15/10/2019	1.5475	1.5475	0	0.413898	0.36225
16/10/2019	0.502495	0.502495	0	0.420226	0.2605
17/10/2019	0.502503	0.502503	0	0.426103	0.209108
18/10/2019	-1.498497	0	1.498497	0.395667	0.316143
21/10/2019	1.407493	1.407493	0	0.46794	0.315215
22/10/2019	-0.996498	0	0.996498	0.434516	0.306321

Figure 4: Example Of Average Gain And Average Loss In The Amazon Stock

Next, using the average gain and average loss, the relative strength was calculated. The relative strength was calculated by using the formula 'Average Gain/ Average Loss'. An example of this is shown is shown below. The RS column refers to the Relative strength.

Date	price change	Gains	Losses	Avg gain	Avg loss	RS
04/10/2019	0.761497	0.761497	0	0.35067	0.537643	0.652237
07/10/2019	-0.349495		0	0.349495	0.325623	0.562607	0.578775
08/10/2019	-1.357506	0	1.357506	0.302364	0.641393	0.471418
09/10/2019	0.824006	0.824006	0	0.339624	0.641393	0.52951
10/10/2019	-0.086503	0	0.086503	0.315365	0.549929	0.573465
11/10/2019	0.583001	0.583001	0	0.334482	0.518286	0.645362
14/10/2019	0.225502	0.225502	0	0.326698	0.36225	0.901856
15/10/2019	1.5475	1.5475	0	0.413898	0.36225	1.142574
16/10/2019	0.502495	0.502495	0	0.420226	0.2605	1.61315
17/10/2019	0.502503	0.502503	0	0.426103	0.209108	2.037721
18/10/2019	-1.498497	0	1.498497	0.395667	0.316143	1.251544
21/10/2019	1.407493	1.407493	0	0.46794	0.315215	1.484514
22/10/2019	-0.996498	0	0.996498	0.434516	0.306321	1.418498

Figure 5: Example Of Relative Strength From Amazon Stock Data

Finally, the relative strength index is calculated using the following formula:
RSI = 100 - (100 / (1 + RS)).

Date	price change	Gains	Losses	Avg gain	Avg loss	RS	RSI	HL
04/10/2019	0.761497	0.761497	0	0.35067	0.537643	0.652237	39.47599	1.067497
07/10/2019	-0.349495	0	0.349495	0.325623	0.562607	0.578775	36.65974	1.206504
08/10/2019	-1.357506	0	1.357506	0.302364	0.641393	0.471418	32.03832	1.099998
09/10/2019	0.824006	0.824006	0	0.339624	0.641393	0.52951	34.61959	0.779496
10/10/2019	-0.086503	0	0.086503	0.315365	0.549929	0.573465	36.446	1.226997
11/10/2019	0.583001	0.583001	0	0.334482	0.518286	0.645362	39.22309	0.779503
14/10/2019	0.225502	0.225502	0	0.326698	0.36225	0.901856	47.41979	0.9945
15/10/2019	1.5475	1.5475	0	0.413898	0.36225	1.142574	53.32717	1.791504
16/10/2019	0.502495	0.502495	0	0.420226	0.2605	1.61315	61.73201	0.785995
17/10/2019	0.502503	0.502503	0	0.426103	0.209108	2.037721	67.08059	0.841499
18/10/2019	-1.498497	0	1.498497	0.395667	0.316143	1.251544	55.58604	2.238998
21/10/2019	1.407493	1.407493	0	0.46794	0.315215	1.484514	59.75069	1.043999
22/10/2019	-0.996498	0	0.996498	0.434516	0.306321	1.418498	58.65202	1.389
23/10/2019	-0.178001	0	0.178001	0.403479	0.319036	1.264684	55.84372	1.402504

Figure 6: Screenshot Of All Derived Features

Machine Learning Models And Code

The following models were used: Linear Regression, Decision Tree, Random Forest, XGBoost and K-nearest Neighbour (KNN). The details of each model are discussed below where necessary along with some common code.

Common Code

```
1  import pandas as pd
2  #ML models
3  from sklearn.ensemble import RandomForestRegressor
4  from sklearn.linear_model import LinearRegression
5  from sklearn.tree import DecisionTreeRegressor
6  from xgboost import XGBRegressor
7  from sklearn.neighbors import KNeighborsRegressor
8  #Split
9  from sklearn.model_selection import train_test_split
10 #Results
11 from sklearn.metrics import mean_squared_error
12 from sklearn.metrics import mean_absolute_error
13 from sklearn.metrics import mean_absolute_percentage_error
14 #Tuning
15 from sklearn.model_selection import GridSearchCV
16
17 filepath = 'TSLA.csv'
18 data = pd.read_csv(filepath)
19 y = data.Close
20 features = ['Volume', 'RSI', 'HL','CPC']
21
22 X = data[features]
23
24 train_X, val_X, train_y, val_y = train_test_split(X, y, test_size=0.2, random_state=0)
```

Figure 7: Screenshot Of Common Code

The above code first imports Pandas, next all the necessary machine learning models. It also imports the train test split feature (3) in line 9. This feature splits the datasets into training and testing sets. This means that some part of the dataset will be used to train the models, and the remaining part will be used for testing the accuracy of the model. Moving on, three types of evaluating metrics are imported: the mean squared error (4), the mean absolute error (5) and the mean absolute percentage error (6). These three metrics are used in order to obtain a comprehensive evaluation of each model. Each metric captures different error characteristics: the mean squared error is sensitive to outliers and hence highlights larger errors; the mean absolute error offers a straightforward estimate of the average prediction error, regardless of outliers. Finally, the mean absolute percentage error allows the error to be interpretable as a percentage, allowing it to be easily understood. Moving on, GridSearchCV (7) was imported to select the best hyperparameters and so improve the performance of all models. Lines 17 and 18 tell the program which stock to analyse, and line 19 sets the close price as a target for prediction. Line 20 selects the features to use as the basis for prediction. Line 24 tells the program how much data should be training data and testing data. I have selected 20% of the data to be testing data and 80% as training data, as generally the majority of the data must be training data for the model to get an accurate understanding of the data. The random state feature is used here and throughout the code in order to make the results replicable.

```
#Linear regression
linear_model = LinearRegression()
linear_model.fit(train_X,train_y)
pred = linear_model.predict(val_X)
```

Figure 8: Linear Regression Code

This is the standard code for fitting a dataset with linear regression (geeksforgeeks, 2024) and making a prediction.

Other Models

All the following models use the grid search feature in order to find the best hyperparameters from the parameter grid. The rest of the code for each model is the standard code to utilise those models.

```
#Decision tree
tree_model = DecisionTreeRegressor(random_state=1)
tree_model.fit(train_X,train_y)
param_grid = {
    'max_depth': [3, 5, 7, 10, 15, None],
    'min_samples_split': [2, 5, 10, 20, 50],
    'min_samples_leaf': [1, 2, 5, 10,50],
  'max_leaf_nodes': [None, 10, 20, 30],}
```

(geeksforgeeks, 2023)

Figure 9: Decision Tree Code

```
#Random forest
forest_model = RandomForestRegressor(random_state=1)
forest_model.fit(train_X, train_y)

param_grid = {
    'n_estimators': [100, 200, 300, 500],
   'max_depth': [None, 10, 20, 30],
     'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],}
```

(Geeksforgeeks, 2023)

Figure 10: Random Forest Code

```
#XGBoost
xgb_model = XGBRegressor()
xgb_model.fit(train_X,train_y)
param_grid = {
  'n_estimators': [50, 100, 200,500,1000],
    'max_depth': [ 5, 7, 10, None],
    'learning_rate': [0.001, 0.01, 0.05, 0.1, 0.3],
    'gamma': [0, 0.1, 0.2, 0.3],}
```

(educative, n.d.)

Figure 11: GBoost Code

```
#KNN
knn_model = KNeighborsRegressor()
param_grid = {
    'n_neighbors': [1,3, 5, 7, 9, 11,15,20,50,100, 200, 300,500,],
    'weights': ['uniform', 'distance'],
    'leaf_size': [10, 20, 30, 40, 50, 60,70, 100],
    'p': [1, 2]}
```

(Korstanje, n.d.)

Figure 12: K Nearest Neighbour Code

Grid Search Code

```
grid_search = GridSearchCV(estimator=knn_model, param_grid=param_grid, cv=5, n_jobs=-1, scoring='neg_mean_absolute_error')
grid_search.fit(train_X,train_y)
best_model = grid_search.best_estimator_
best_params = grid_search.best_params_
pred = best_model.predict(val_X)
```

Figure 13: Grid Search code

Firstly, in the above code, the parameters of GridSearchCV are set. The estimator is set to the model that is currently being used. The parameter grid is set to the parameter grid used and 'cv' is set to 5 because it is considered to be a balance between computational efficiency and quality evaluation. N jobs is set to -1 to allow parallel processing and using all cores of the computer, leading to faster processing. Negative mean absolute error is used as the scoring parameter so that GridSearchCV selects hyperparameters that aim to reduce the mean absolute error, improving prediction accuracy. The code from line 68 to 70 is the standard code for GridSearchCV.

Printing Results

```
72  print(best_params)
73  print('mae=', mean_absolute_error(val_y, pred))
74  print('mse=', mean_squared_error(val_y, pred))
75  print('mape=', mean_absolute_percentage_error(val_y,pred)*100)
```

Figure 14: Printing Results Code

The code in line 72 prints the best parameters according to GridSearchCV for each model in use. The rest of the code from line 73 to 74 prints the results of the model's evaluation metrics.

```
{'max_depth': 7, 'max_leaf_nodes': 30, 'min_samples_leaf': 1, 'min_samples_split': 2}
mae= 14.053909675940751
mse= 402.7148230360385
mape= 11.612834792379465
```

Figure 15: Example Results

The above figure shows the result for predicting the Apple stock using the Decision tree model. The first line outputs the best hyperparameters and the next three lines output the evaluation metrics for this model.

Best Hyperparameters For Each Model

Hyperparameters were available for all used models other than Linear regression. The hyperparameter settings for each model shown below for each company's dataset shows the parameters that lead to best the result according to the negative absolute error metric.

Table – 1 Decision Tree Hyperparameters

	Max depth	Max leaf nodes	Min samples leaf	Min samples split
Amazon	7	10	50	2
Apple	7	30	1	2
Meta	15	None	2	50
Tesla	10	None	5	2
Nvidia	7	None	5	2

Table-2 Random Forest Hyperparameters

	Max depth	Max leaf nodes	Min samples leaf	Min samples split
Amazon	10	1	10	300
Apple	None	1	2	200
Meta	10	2	10	500
Tesla	10	2	2	500
Nvidia	10	1	2	500

Table-3 XGboost Hyperparameters

	Gamma	Learning rate	Max depth	N estimators
Amazon	0	0.01	5	200
Apple	0.2	0.1	None	500
Meta	0.2	0.1	5	1000
Tesla	0.1	0.1	5	50
Nvidia	0	0.1	5	500

Table-4 K-nearest Neighbour

	Leaf size	N neighbours	P	Weights
Amazon	10	20	1	uniform
Apple	10	100	1	uniform
Meta	10	100	1	uniform
Tesla	10	100	1	uniform
Nvidia	10	100	1	uniform

RESULTS

The best model for each metric for is bolded. The following are the results of this investigation:

Table – 5 Amazon Results

Metrics	Amazon				
	Linear regression	Decision tree	Random Forest	XGBoost	KNN
MAE	24.4225	22.4746	**21.2326**	22.1581	26.0148
MSE	774.1687	742.0268	**664.1475**	664.92	889.67
MAPE	19.06128	17.4435	**16.4488**	17.2055	20.9471

Table – 6 Apple Results

Metrics	Apple				
	Linear regression	Decision tree	Random Forest	XGBoost	KNN
MAE	21.2004	14.0539	**12.3301**	12.4383	24.1099
MSE	742.3418	402.7148	**243.9947**	253.178	1066.38
MAPE	20.8039	11.6128	**9.9406**	10.324	25.9248

Table – 6 Meta Results

Metrics	Meta				
	Linear regression	Decision tree	Random Forest	XGBoost	KNN
MAE	55.2351	46.7314	**39.8791**	39.9256	67.9091
MSE	4684.9978	3971.8018	2888.5663	**2815.52**	7289.42
MAPE	23.5328	19.6783	**16.9451**	17.0958	29.3374

Table – 7 Tesla Results

Metrics	Tesla				
	Linear regression	Decision tree	Random Forest	XGBoost	KNN
MAE	43.3039	27.1488	**22.5718**	22.6034	62.4008
MSE	2991.6772	1423.4245	891.6114	**875.4**	6541.18
MAPE	75.2704	15.8931	**13.2676**	14.3861	134.029

Table – 8 Nvidia Results

Metrics	Nvidia				
	Linear regression	Decision tree	Random Forest	XGBoost	KNN
MAE	7.2571	5.8787	**5.2307**	5.2401	15.7301
MSE	113.5031	89.7645	63.8089	**61.8701**	527.488
MAPE	40.3879	23.9811	**21.841**	23.1061	106.067

Overall, the random forest model worked best for all companies, with the exception being XGboost in the case of Meta, Tesla and Nvidia for the Mean squared error metric.

CONCLUSION

It can be concluded that the random forest model works best among the models selected for these companies. However, currently the model lacks accuracy due to the small number of features used. This research can be extended to include many more features such as the number of X (formerly twitter) mentions, positive and negative news mentions, general economic indicators amongst others. This, along with increasing the hyperparameters tested will drastically improve the accuracy of the model and make it much more worthwhile for investors to use the model as a reference. For the extension of this research, the random forest model is highly suitable as it emerged as the best model amongst models tested in this research.

REFERENCES

1. educative. (n.d.). Regression using XGBoost in Python. Retrieved June 22, 2024, from educative.io: https://www.educative.io/answers/regression-using-xgboost-in-python
2. geeksforgeeks. (2023, January 11). Python | Decision Tree Regression using sklearn. Retrieved June 20, 2024, from geeksforgeeks: https://www.geeksforgeeks.org/python-decision-tree-regression-using-sklearn/
3. Geeksforgeeks. (2023, December 16). Random Forest Regression in Python. Retrieved June 20, 2024, from geeksforgeeks.org: https://www.geeksforgeeks.org/random-forest-regression-in-python/
4. geeksforgeeks. (2024, May 22). Python | Linear Regression using sklearn. Retrieved June 20, 2024, from Geeksforgeeks: https://www. geeksforgeeks.org/python-linear-regression-using-sklearn/
5. Geeksforgeeks. (n.d.). How To Do Train Test Split Using Sklearn In Python. Retrieved June 15, 2024, from geeksforgeeks.org: https://www.geeksforgeeks.org/how-to-do-train-test-split-using-sklearn-in-python/
5. howtoexcel. (n.d.). How to Calculate RSI in Excel. Retrieved June 15, 2024, from howtoexcel.net: https://howtoexcel.net/2023/05/how-to-calculate-rsi-in-excel.html
6. Korstanje, J. (n.d.). The k-Nearest Neighbors (kNN) Algorithm in Python. Retrieved June 22, 2024, from realpython.com: https://realpython.com/knn-python/
7. Remesh, A. (n.d.). The Advantages and Disadvantages of Technical Analysis. Retrieved June 13, 2024, from strike.money: https://www.strike.money/technical-analysis/pros-and-cons
8. scikit-learn. (n.d.). mean_absolute_error. Retrieved June 16, 2024, from scikit-learn.org: https://scikit-learn.org/stable/modules/ generated/sklearn.metrics.mean_absolute_error.html
9. scikit-learn. (n.d.). mean_absolute_percentage_error. Retrieved June 16, 2024, from scikit-learn.org: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_absolute_percentage_error.html
10. scikit-learn. (n.d.). mean_squared_error. Retrieved June 15, 2024, from scikit-learn.org: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_squared_error.html
11. Verma, A. (2023, February 10). GridSearchCV in scikit-learn: A Comprehensive Guide. Retrieved June 20, 2024, from dev.to: https://dev.to/anurag629/gridsearchcv-in-scikit-learn-a-comprehensive-guide-2a72