Stock Performance Predictor

Kaushal Patil
kaushal.p@ahduni.edu.in

Arpitsinh Vaghela
arpitsinh.v@ahduni.edu.in

Prachee Javiya
prachee.j@ahduni.edu.in

Vrunda Gadesha
vrunda.g@ahduni.edu.in

***Index Terms*—Financial Machine Learning, Quant Trading, Classification/Clustering, Boosting**

***Abstract*—This project aims to study and test fundamental and technical features to create a model that can classify/-cluster stock tickers as BUY SELL or HOLD signals. Using algorithms such as Logistic Regression, Linear Regression, K-nearest Neighbor, Decision Trees to create an accurate classifier/clustering model and use Bagging and Boosting techniques to achieve better performance.**

I. Introduction

Predicting the stock price trend by interpreting the seemly chaotic market data has always been an attractive topic to both investors and researchers. When it comes to trading in the Stock Market, there are many different approaches to find the right stock. Many forms of analysis have emerged to detect which stock is worth the money. Technical Analysis, Sentimental Analysis and Fundamental Analysis — observing a company's financials like their Balance Sheet or Cash Flow Statement to determine if the company has value relative to their current stock price. These are not the only forms of stock analysis out there but they are arguably the most popular. Among those popular methods that have been employed, Machine Learning techniques are very popular due to the capacity of identifying stock trend from massive amounts of data that capture the underlying stock price dynamics. In this project, we have applied numerous machine learning techniques to build a model for stock classification - BUY, SELL or HOLD.

II. Literature Review

Stock market decisions have a huge impact on investors[1]. It is difficult to analyse every aspect manually in order to make the best prediction. There have been numerous researches in this field. [2] is based on the approach of predicting the share price using Long Short Term Memory (LSTM) and Recurrent Neural Networks (RNN) to predict the stock price on NSE data using various factors such as current market price, price-earning ratio, base value and some miscellaneous events. In order to achieve higher accuracy, deep learning methods are generally preferred - owing to the fact that it builds networks to learn on its own. [3] proposes a deep neural network-based corporate performance prediction model that uses a company's financial and patent indicators as predictors. The proposed model includes an unsupervised learning phase and a fine-tuning phase.

III. Implementation

A. Data Collection

The training data used was collected from US stock fundamentals database. The data consists of 63,547 entries and 30 fields. The data is divided into 4 quarters for each fiscal year.

	Ticker	SimFinId	Currency	Fiscal Year	Fiscal Period	Report Date	Publish Date	Restated Date	Shares (Basic)
0	A	45846	USD	2010	Q3	2010-07-31	2010-10-06	2010-10-06	347000000.0
1	A	45846	USD	2010	Q4	2010-10-31	2010-12-20	2011-12-16	344000000.0
2	A	45846	USD	2011	Q1	2011-01-31	2011-03-09	2011-03-09	347000000.0
3	A	45846	USD	2011	Q2	2011-04-30	2011-06-07	2011-06-07	347000000.0
4	A	45846	USD	2011	Q3	2011-07-31	2011-09-07	2011-09-07	348000000.0

Fig. 1: Dataset

B. Data Cleaning and pre-processing

The entries in the raw dataset that contained "bad" values were set to 0. The columns which did not contribute in learning were dropped.

```
clean_fundamen2=clean_fundamen.dropna(subset
    =['Price_Future','Price_Present'])
clean_fundamen=clean_fundamen.fillna(0)
```

For Data Visualization, correlation heatmap was plotted:

C. Data Labelling

The data entries were classified into 3 labels, 0,1 and 2 for Buy, Sell or Hold respectively. Present price and future price are compared for every quarter with a difference tolerance of 3%. Three additional columns - Price present, price future and labels are added into the dataset.

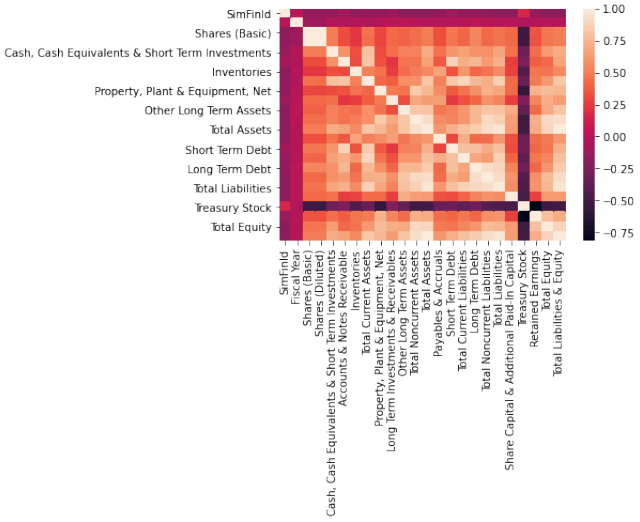

Fig. 2: Correlation heatmap using Seaborn

	Ticker	Price_Present	Price_Future	Label
0	A	32.67	41.48	0
1	A	40.63	43.95	0
2	A	46.05	48.71	0
3	A	47.7	35.69	1
4	A	35.69	36.66	2
5	A	33.46	45.62	0

Fig. 3: Labelled dataset

D. Algorithms

Open source libraries used - Pandas[4], Matplotlib[5] Numpy[6], Seaborn[7] and, Scikit Learn[8]

1) Principal Component Analysis: We performed 2 component PCA and plotted colorcoded datapoints based on their classes to see how the data is distributed. The plot is shown under results.

2) Model Stacking: This algorithm involves building a meta-model based on prediction outputs of other different models. In our implementation, we have stacked lasso, random forest and gradient boosting algorithms. Metrics for each algorithm:
Logistic Regression - Max iteration = 1000
Random Forest - max depth 18, step 1
Gradient Boosting - max depth 18, step 1

3) XG Boosting: XGBoost is a decision-tree-based ensemble Machine Learning algorithm that uses a gradient boosting framework. We have used XGBClassifier library and have applied this on numeric fields in our dataset. Input values are standardized using StandardScaler library. Test train split - 0.1

[74]:

	principal component 1	principal component 2	target_numeric	target
0	0.210449	0.043768	0	Buy
1	0.267291	0.050526	0	Buy
2	0.071494	-0.233925	0	Buy
3	0.126114	-0.227153	1	Sell
4	0.146986	-0.243517	2	Hold
...	...	...	...	...
58823	-1.039740	0.078976	1	Sell
58824	-1.039956	0.079430	0	Buy
58825	-1.037820	0.079236	0	Buy
58826	-1.034459	0.079764	0	Buy
58827	-1.028980	0.081075	0	Buy

58828 rows × 5 columns

Fig. 4: 2 component PCA target values

4) Decision trees: Decision trees is a type of supervised learning algorithm where the data is continuously split according to a certain parameter. The tree can be explained by two entities, namely decision nodes and leaves.
We have tested our dataset with ensemble methods as well as writing a decision tree from scratch. Adding a 4th label, "I don't know" for nodes that resulted in overfitting (impurity <1). We also attempted to kill the nodes which resulted in underfitting and overfitting. Doing so showed some increase in the accuracy.

```
for i in range(len(tree.impurity)):
    if tree.impurity[i]<1:
        tree.value[i][0][3]=max(tree.value[i
            ][0])+10
```

```
0.4476457589665137
[[0]
 [0]
 [1]
 ...
 [0]
 [1]
 [0]]
[1.50058298 1.48111189 1.42527407 ... 0.         0.         0.        ]
Accuracy with some nodes killed:  0.476216814159292
```

Fig. 5: Accuracy difference after killing nodes that were overfitting the data

5) Random Forest Classifier: RandomForestClassifier model is used from Scikit learn library. We reach upto a depth of 32, with 2 steps.

Random forest is an ensemble of decision trees. We have compared the results from three approaches - Using RFC from scikit learn, writing the algorithm from scratch and using AutoML. The results of AutoML showed that Random

```
Max_Depth:  2  Score Train: 0.4718120464809459
Max_Depth:  2  Score Test: 0.4696095601112599
Max_Depth:  4  Score Train: 0.4781803420104531
Max_Depth:  4  Score Test: 0.4746574636859998
Max_Depth:  6  Score Train: 0.4894707464352768
Max_Depth:  6  Score Test: 0.4784691459771299
Max_Depth:  8  Score Train: 0.5114172629015071
Max_Depth:  8  Score Test: 0.48192026372720714
Max_Depth:  10  Score Train: 0.5507941340640382
Max_Depth:  10  Score Test: 0.4859379828989389
```

Fig. 6: Train and test scores for RFC

Forest is the best algorithm for our data. This was compared by the algorithm written from scratch - and the results were similar.

IV. Results

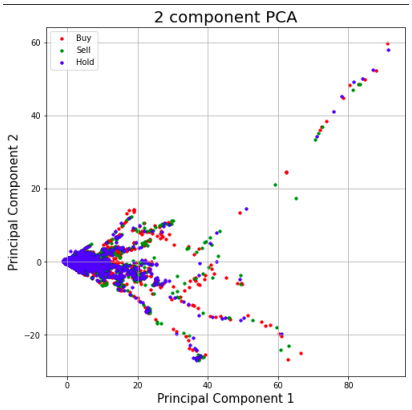

Fig. 7: A clear trend seperating the three labels is not apparent when visualizing it with PCA(components=2)

Random Forest Classifier

At depth = 16, we get the maximum test accuracy (out of 30)

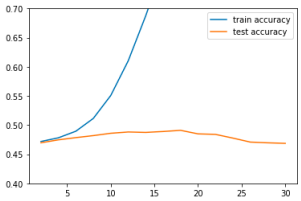

Fig. 8: Train and test accuracy graph for random forest classifier

Auto-sklearn[9] frees a machine learning user from algorithm selection and hyperparameter tuning. It leverages recent

```
Max_Depth: 2 :   Train Accuracy:0.47    Test Accurary:0.472
Max_Depth: 4 :   Train Accuracy:0.477   Test Accurary:0.476
Max_Depth: 6 :   Train Accuracy:0.488   Test Accurary:0.48
Max_Depth: 8 :   Train Accuracy:0.506   Test Accurary:0.484
Max_Depth: 10 :  Train Accuracy:0.541   Test Accurary:0.487
Max_Depth: 12 :  Train Accuracy:0.596   Test Accurary:0.489
Max_Depth: 14 :  Train Accuracy:0.67    Test Accurary:0.492
Max_Depth: 16 :  Train Accuracy:0.756   Test Accurary:0.499
Max_Depth: 18 :  Train Accuracy:0.838   Test Accurary:0.493
Max_Depth: 20 :  Train Accuracy:0.91    Test Accurary:0.487
Max_Depth: 22 :  Train Accuracy:0.961   Test Accurary:0.482
Max_Depth: 24 :  Train Accuracy:0.987   Test Accurary:0.478
Max_Depth: 26 :  Train Accuracy:0.996   Test Accurary:0.477
Max_Depth: 28 :  Train Accuracy:0.999   Test Accurary:0.469
Max_Depth: 30 :  Train Accuracy:1.0     Test Accurary:0.471
```

Fig. 9: Maximum accuracy(used direct model) of 49.9%

```
This Classification Tree Prediction Accuracy:    0.4844467108618052
```

Fig. 10: Custom random forest accuracy - 48.4%

advantages in Bayesian optimization, meta-learning and ensemble construction. We have used AutoML to compare the results and finally get an insight into which algorithm works best for our problem.

```
'classifier:__choice__': 'random_forest',
```

Fig. 11: Output of AutoML, best model - Random forest classifier

1) Evaluation Metrics: In our case, the best performing model came from the XGBoost Classifier and Random Forest. The classification report is as shown:

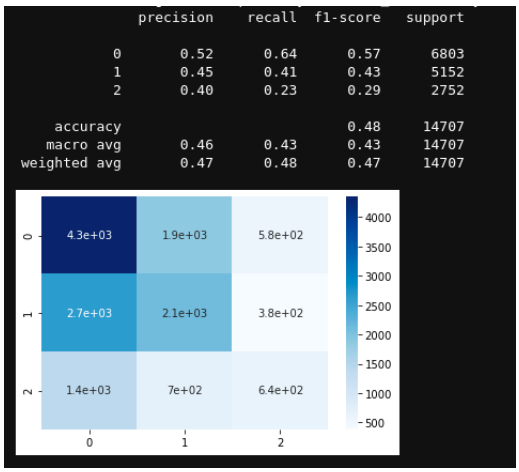

Fig. 12: Evaluation Metrics - XGBoost

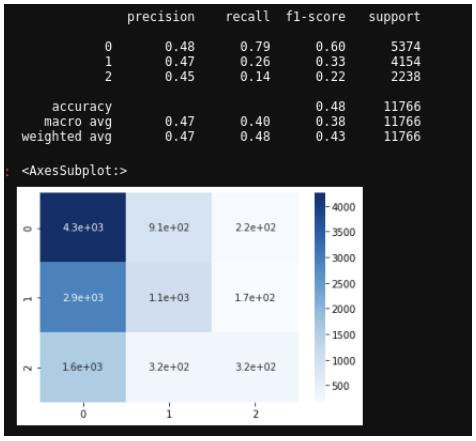

Fig. 13: Evaluation Metrics - Custom Random Forest Classifier

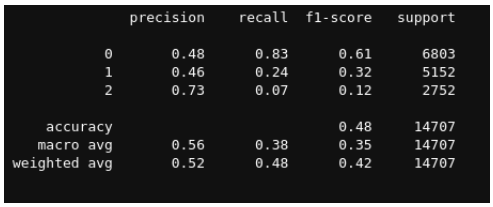

Fig. 14: Evaluation Metrics - Auto Sklearn

V. Conclusion

Deciding a stock's worth based on Quarterly Reports is not a new achievement. For most Fundamental Analysts, the strategy we utilized may be considered too simple since it works successfully 50% of the time. But for the purposes of learning machine learning classification, it was enough. We could potentially add more features to improve the model or alter the features we have based on new strategies. Ripping the decision trees and building algorithms from scratch increased the accuracy. Killing nodes that lead to overfitting also increased the accuracy by 0.2. We saw that out of all the algorithms, Random Forest Classifier and XGBoosting worked the best, giving 50% accuracy.

References

[1] A. S. G. P. . Kohli P.P.S., Zargar S., "Stock prediction using machine learning algorithms. in: Malik h., srivastava s., sood y., ahmad a. (eds) applications of artificial intelligence techniques in engineering." [Online]. Available: https://doi.org/10.1007/978-981-13-1819-1$_38$

[2] B. Jeevan, E. Naresh, B. P. V. kumar, and P. Kambli, "Share price prediction using machine learning technique," pp. 1–4, 2018.

[3] J. Lee, D. Jang, and S. Park, "Deep learning-based corporate performance prediction model considering technical capability," *Sustainability*, vol. 9, no. 6, 2017. [Online]. Available: https://www.mdpi.com/2071-1050/9/6/899

[4] T. pandas development team, "pandas-dev/pandas: Pandas," Feb. 2020. [Online]. Available: https://doi.org/10.5281/zenodo.3509134

[5] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[6] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del R'ıo, M. Wiebe, P. Peterson, P. G'erard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2

[7] M. L. Waskom, "seaborn: statistical data visualization," *Journal of Open Source Software*, vol. 6, no. 60, p. 3021, 2021. [Online]. Available: https://doi.org/10.21105/joss.03021

[8] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[9] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter, "Efficient and robust automated machine learning," in *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds. Curran Associates, Inc., 2015, pp. 2962–2970. [Online]. Available: http://papers.nips.cc/paper/5872-efficient-and-robust-automated-machine-learning.pdf