

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 seemingly 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

Libraries used : Matplotlib, Pandas, Seaborn, Scikit learn and numpy

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.

```
1 clean_fundamen2=clean_fundamen.dropna(subset=[ 'Price_Future', 'Price_Present' ])
2 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.

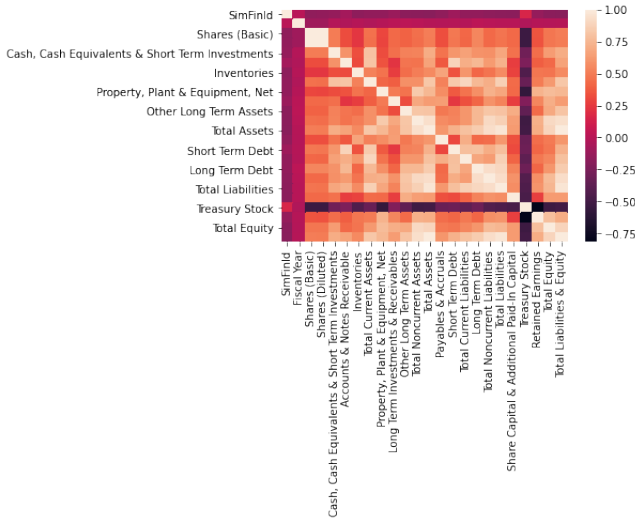


Fig. 2: Correlation heatmap using Seaborn

	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], Numpy[5], seaborn[6] and, scikit learn[7]

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) *K means clustering with 2 component PCA*: Partitioning n observations into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster.

3) *K nearest Neighbours*: The KNN algorithm assumes that similar things exist in close proximity. For stock performance prediction KNN-Classification is suitable. We found MSE using KNN Algorithm and defined elbow function to find the appropriate value of K.

[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

Fig. 4: 2 component PCA target values

4) *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

5) *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

6) *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.

```

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

```

7) *Random Forest Classifier*: RandomForestClassifier model is used from Scikit learn library. We reach up to 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. 5: 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

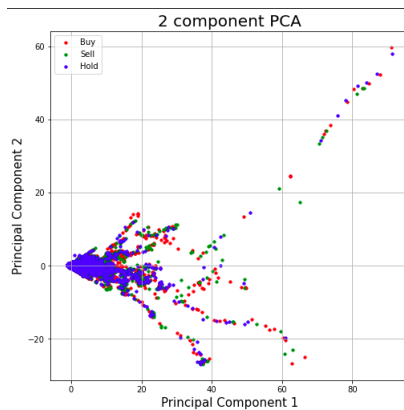


Fig. 6: A clear trend separating the three labels is not apparent when visualizing it with PCA(components=2)

Random Forest Classifier

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

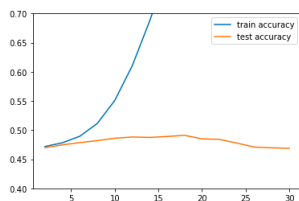


Fig. 7: Train and test accuracy graph for random forest classifier

Auto-sklearn[8] frees a machine learning user from algorithm selection and hyperparameter tuning. It leverages recent

```

for i in range(2,32,2):
    clf = RandomForestClassifier( max_depth=i,random_state=0)
    # Full overfit
    clf.fit(Xtrain, Ytrain)
    trainscore=clf.score(Xtrain,Ytrain)
    trainscores.append(trainscore)
    testscore=clf.score(Xtest,Ytest)
    testscores.append(testscore)
    print("Max_Depth: ",i," Score Train:",trainscore)
    print("Max_Depth: ",i," Score Test:",testscore)

```

```

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: 0.4747089729061502
Max_Depth: 6 Score Train: 0.48924240117724666
Max_Depth: 6 Score Test: 0.4784691459771299
Max_Depth: 8 Score Train: 0.5107068554320799
Max_Depth: 8 Score Test: 0.48228082826826
Max_Depth: 10 Score Train: 0.5495509209925408
Max_Depth: 10 Score Test: 0.486710621201195
Max_Depth: 12 Score Train: 0.6126756989902066
Max_Depth: 12 Score Test: 0.48779231482435353
Max_Depth: 14 Score Train: 0.6870147663266859
Max_Depth: 14 Score Test: 0.49150097867518283
Max_Depth: 16 Score Train: 0.775282894403004
Max_Depth: 16 Score Test: 0.4924796538580406
Max_Depth: 18 Score Train: 0.8615720302430608
Max_Depth: 18 Score Test: 0.4917070155557845
Max_Depth: 20 Score Train: 0.9235043385599025
Max_Depth: 20 Score Test: 0.48619552899969093
Max_Depth: 22 Score Train: 0.9660780433348556
Max_Depth: 22 Score Test: 0.48228082826826
Max_Depth: 24 Score Train: 0.9890140559192165
Max_Depth: 24 Score Test: 0.4762542495106624
Max_Depth: 26 Score Train: 0.9963464758715177
Max_Depth: 26 Score Test: 0.47455444524569895
Max_Depth: 28 Score Train: 0.9993657076165829
Max_Depth: 28 Score Test: 0.4703306891933656
Max_Depth: 30 Score Train: 0.99984776982798
Max_Depth: 30 Score Test: 0.46893994024930463

```

Fig. 8: Maximum accuracy(used direct model) of 49.2%

```

This Classification Tree Prediction Accuracy: 0.4844467108618052

```

Fig. 9: 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. 10: 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:

	precision	recall	f1-score	support
0	0.48	0.83	0.61	6803
1	0.46	0.24	0.32	5152
2	0.73	0.07	0.12	2752
accuracy			0.48	14707
macro avg	0.56	0.38	0.35	14707
weighted avg	0.52	0.48	0.42	14707

Fig. 11: Evaluation Metrics

2015, pp. 2962–2970. [Online]. Available: <http://papers.nips.cc/paper/5872-efficient-and-robust-automated-machine-learning.pdf>

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 or outdated. 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.

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] 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'io, M. Wiebe, P. Peterson, P. G'erald-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>
- [6] 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>
- [7] 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.
- [8] M. Feurer, A. Klein, K. Eggenberger, 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.,