University of Zurich^UZH

Machine Learning in Finance

Group Project Summary

Department of Banking and Finance

Spring Semester 2021

Seminar Introduction to Machine Learning

Aaron Hauser

Lorena Tassone

Lukas Dekker

Nick Vogel

Missing data

Since we've found that there are lots of missing values, we spent some time researching how to tackle this issue. Our research came up with some papers, as well as blog entries surrounding the topic from which we took inspiration – some of which are listed in the appendix. Albeit some of the proposed methods were too involved for our understanding and the scope of this project we've come up with some methods which seemed worth considering.

Some basic methods include mean, median or most frequent value (mode) but these approaches seemed to be rather simplistic for our purposes. Therefore, we settled for more elaborate measures, such as KNN, MissForest and iterative imputer.

Before we got started with implementing these methods, we conducted some EDA. One thing that caught our eye was the fact that some columns had a ton of NAs. Since imputing, with whatever method, does not seem reasonable for such factors we had to decide which ones we wanted to drop. Eventually, we settled for a benchmark of 1'500 NAs per column (out of roughly 4000-5000 values per year) and year.

After finding these high-NA variables we wanted to check whether some of them could be useful to assign stocks to one of the predefined labels (buy/hold/sell). We achieved this by creating a classification tree. Since at that point of time we were not actually going to make a prediction based on the results, but were only concerned with finding factors that may be 'important', we used the predefined variable 'Class' as our response variable. From this we got eight variables which were used in a decision tree for any of the years, which had a high NA-count. We then looked at each of them individually to determine whether to drop them or not.

What is more, we have found that there were also a number of rows which had a rather high NA-count. For this we've used a threshold of 150 missing values per row, which accounted for approximately 10% of all firms (where a company is counted twice if it appears in two years, three times if it appears thrice, etc.). Any row with a higher NA-count was dropped. We were able to see a clear improvement regarding the total NA-count.

Zeros

Another problem that had to be addressed were the zero-values found throughout the data set. This seemed to be a rather delicate matter, since we were and still cannot be sure whether these zero-values represent actual zeroes, NAs or something else which we weren't considering.

After some plotting and further exploration of the data it showed that a lot of zeroes were connected to either dividends, R&D or things like Revenue from discontinued Operations, short or long term investments which are metrics that are not unlikely to be zeroes if seen in an economical sense (see appendix for a graph). Eventually, we decided to leave the zero-values untouched, by the reasoning that we would possibly do more harm than good by changing these values, as is suggested by various literature.

Outliers

A next step in approaching our main task was the handling of outliers, which we met with a method that is widely used according to scientific literature: the IQR-approach. As an alternative we've looked at the Winsorization method to deal with the outliers which is very close to the IQR-approach.

Some of the problems that we've encountered for this part of the project are firstly, the fact that a lot of methods assume normality which is violated for the given data as we've checked with the Henze-Zirkler-test. Secondly a lot of the literature is concerned with univariate data which is obviously not given either so finding methods that can be applied for multivariate data was rather challenging.

Again, one might add that this might bring more misinformation than good to the table so another way would be to simply leave the outliers alone. As one user has mentioned on the Kaggle webpage some values are in a wrong scale e.g., revenue has not been converted to USD but left in Yen.

Feature Engineering & Selection

Before selecting the essential features, we needed to create a dummy variable for every sector. Furthermore, we decided to create dummy variables for the market cap. We assigned each stock to one of the three groups: small, mid, or large-cap. The selection was based on the definitions from Barone, A. (2020). In order to increase the quality of the dataset, we decided to create some new features. As Gu, S., Kelly, B., & Xiu, D. (2018) showed, one of the most critical features for stock return prediction is the momentum factor. This factor represents the past performance of a given stock. We decided to take the performance of year t-1 in order to predict the performance of year t. We also had the idea to create an interaction term using a macroeconomic factor. According to Zucchi, K. (2021), value stocks perform better in high inflation periods while growth stocks perform better during low inflation periods. We tried to add this effect into our models. While Value Stocks are characterized by a high dividend yield, a low P/B ratio, and a low P/E ratio, growth stocks are characterized by a high P/E Ratio and typically pay no dividend (Smith, T. (2020)). We therefore created interaction terms with the inflation and the dividend yield, the P/B ratio, and the P/E ratio.

Since our original data set has 222 financial features, we are dealing with a rather high dimensional set. Therefore, a pre-processing method that identifies the most important features can enhance the predictive accuracy (John, Kohavi, Pfleger (1994)) and improve both the computational and the general performance of our learning algorithm (Shukla et al. (2020)).

For selecting the most important features, we decided to use the Random Forest algorithm, which ranks the features importance based on an impurity decrease (Pedregosa et al. (2011)).

To cover most of the importance, we decided to select all the features whose importance is higher or equal the median of all ranked importance values which resulted in a cumulated importance of 67.7%. However, of the 64 by the algorithm selected features (see appendix figure 3) we can see that only the first two features are significantly higher. Additionally, it is interesting to see, that the first three indicators those are, which were added to the dataset by us.

Before the final pipeline selects these features, we removed constant features that have the same value for all stocks. Additionally, we removed one of two correlated features with a correlation coefficient higher than 0.8 since they convey redundant information to the model. The threshold 0.8 is still quite high but setting it lower would lead to a removal of too many features. Further, there was no significant improvement of performance when setting this threshold lower. The removal of these features led to an increase of the feature's importance values.

When testing some models without feature selection, there was no significant improvement of performance. Only Random Forest's performance was slightly better. This was not the case for SVM and some others, which is why we decided to use the selection for the remaining classifiers.

Class Imbalance

Since the "hold" classification is defined by a small range (within +/- 2.5% of S&P500), the original amount of "hold" recommendations was way lower than "buy" and "sell" (see appendix figure 4). Therefore, we sampled the class "hold" up by taking 8000 more samples. With "sell" having 10'092 and “buy” 8970 classification results, 8000 seemed to be accurate since it is still lower then the other two but more or less in the same range.

When fitting the models, we additionally checked whether the balanced data set leads to better train and test scores and performance in general than the unbalanced. This was proven true by nearly all classifiers.

Classifiers

Decision Tree/Random Forest

The tuning of the parameters for Random Forest resulted in this final selection (from randomized parameter search):
criterion: ‘entropy’, max_depth: 9, min_sample_leaf: 5, min_samples_split: 15 (for unbalanced data)
criterion: ‘entropy’, max_depth: 13, min_sample_leaf: 2, min_samples_split: 5 (for balanced data)
This led to a test score of 0.6445 for unbalanced and 0.7587 for balanced data. Additionally we’ve fitted a Random Forest model with the default values (criterion: ‘gini’, max_depth: none, min_sample_leaf: 1, min_samples_split: 2) where we see that the test accuracy (unbalanced: 0.6440, balanced: 0.7523) is very close to one obtained with the model employed where we’ve used hyperparameter tuning.

Gradient Boosting Classifier

The tuning of the parameters for Gradient Boosting Classifier resulted in this final selection:
Loss: ‘deviance’, criterion: ‘mse’, max_depth: 10, n_estimators: 300 (for both unbalanced and balanced data).
This led to a test score of 0.6478 for unbalanced and 0.7630 for balanced data. The train score was both times 100% and therefore, we may have some overfitting. This model had the best score of all the models.

Logistic Regression

The tuning of the parameters for Logistic Regression Classifier resulted in this final selection:
Penalty: ‘l2’, C: 1.0, solver: ‘saga’ (for balanced and unbalanced data). We had to use multi_class = multinomial because we had more than two groups.
This led to a test score of 0.4519 for unbalanced and 0.4240 for balanced data. Interesting is the fact that this model worked better for the unbalanced data. Overall this model performed the worst.

Linear Discriminant Analysis

The tuning of the parameters for Linear Discriminant Classifier resulted in this final selection:
Solver: ‘eigen’, shrinkage: ‘auto’ (for balanced and unbalanced data). This led to a test score of 0.5925 for unbalanced and 0.4318 for balanced data. We tried to improve the performance of the model using bagging. Unfortunately, bagging was not able to improve the performance of the model significantly.

Quadratic Discriminant Analysis

The tuning of the parameters for Quadratic Discriminant Classifier resulted in this final selection:
tol: 0.00001 for unbalanced data and tol: 0.0001 for balanced data. This led to a test score of 0.5272 for the unbalanced data and 0.4791 for the balanced data. As already seen with the LDA model, we tried to improve the performance of the models via bagging. We improved the performance for the balanced data slightly, but the model still performed very poorly. Interesting is, that QDA performed better than LDA on the balanced data while LDA performed better on the unbalanced data.

SVM

The tuning of the parameters for SVM (computationally and manually) resulted in this final selection:
kernel: ‘rbf’, C: 0.8, gamma: 0.06. This led to a train score of 0.9437 and a test score of 0.7254. The hyperparameter tuning preferred a C value of 1, however, we chose to try smaller C values, since

a high C value aims to classify all training samples perfectly (Scikit-learn (2011)) and we had a high overfitting problem. Lowering C reduced the overfitting but also lowered the test and train score. Finally, a C of 0.8 was chosen since by looking at the amount of missed 'sell' classifications this seemed to be the most reasonable compromise.

KNN

The tuning of the parameters for KNN (computationally and manually) resulted in this final selection: leaf_size: 100, n_neighbors: 5, p: 2, weights: 'uniform'. This led to a train score of 0.7468 and a test score of 0.6321.

When testing KNN higher leaf_size values and smaller n_neighbors (K) values led to better train and test scores. However, the difference between train and test score was significant since in some cases we generated a train score of 1.000. This was an obvious sign of overfitting which is why a higher n_neighbors value was used. Obviously, the train and test score decreased but the difference between them decreased as well. It was not easy to find a small enough K to avoid oversimplifying but also a large enough to not overfit the samples. Further, using the bagging method did not improve the model

Multilayer Perceptron

The tuning of the parameters for Multilayer Perceptron Classifier first resulted in this final selection: Activation: 'relu', solver: 'lbfgs', max_iter: 1000, alpha: 0.0001, max_fun: 5000 (for unbalanced and balanced data). Because of high overfitting we manually increased alpha (a penalty for variance) and realized that 0.0001 was only a local maximum, and the performance was optimal with alpha: 5.0. This led to a test score of 0.6916. To handle the overfitting, we decided to bag the model and achieved a test score of 0.75.

Results

Finally, with a test score of 0.7632, the Gradient Boosting Classifier achieved the best performance (see appendix figure 5).

Looking at its confusion matrix (see appendix figure 6) we see that our best model performs well for buy, perfectly for hold and medium for sell worthy stocks. For this kind of classification task where the model supports investment recommendations, a low misclassification rate of the class "sell" is crucial. Our model misclassifies 42 out of 100 sell worthy stocks which is quite bad.

Since the difference between train (1.0000) and test score (0.7632) is significant (overfitting) and additionally the test score is not that high, we are not too convinced of our result with Gradient Boosting.

Our models can add value compared to just holding all the stocks since we had an accuracy of over 76%. Nevertheless, there are some points that one has to be aware of before using the models. First of all, it may be possible that the investment strategy's performance using these models may not be better than just holding all the stocks because the misclassified stocks may perform above-average bad or good. Furthermore, we have to be aware that the models are built with historical data, and it is always possible that the market changes and a strategy won't work in the future. Finally, one also has to take the transaction costs of such a strategy into account since the portfolio needs to be rearranged every month.

Difficulties and Problems

- One of the main problems that we have encountered throughout the project was the lacking processing power, i.e., at times we have spent quite some time waiting for python to execute code. This was especially a problem for the imputation of missing values and the hyperparameter tuning when fitting the models. It was not possible to tune a lot of values for each parameter at once, since this took several hours. Therefore, some steps were performed manually.
- Although we all had some experience in coding, the adaption to Jupyter Notebooks needed some time, especially for the handling of import statements, which sometimes had to be imported via the terminal first before we were able to access it in the Notebook
- We frequently used the documentation of different tools and functions to get a more in-depth understanding of functions and their respective parameters, this however, often was quite complex.
- One problem that did not have a big impact but was still somewhat annoying is the labelling of plots. It seemed as though sometimes the same settings gave different results with respect to the size of the axis labels.
- When fitting the models, we often came across the problem of overfitting the train data. One way to solve this was bagging the classifier, however, this did not work for all models. Another way was to vary the parameters manually, for example lowering the parameter C for SVM or increase the parameter n_neighbors (K) for KNN (whose optimal value was previously chosen by grid search). The difficulty was here to find a low but at the same time high enough value to neither oversimplify nor overfit the data.

Further Actions

If we had more resources (mainly time and processing power) some further steps that we would take are the following:

- We have created different datasets, one for each different imputing method, and an additional one for the Winsorization method (see chapter on outliers), but we've only fitted our models on the dataset obtained via KNN imputer, and the IQR-method, as a future step we'd also fit the models on the other datasets (i.e., other imputing and outlier-handling methods)
- Create new features, complementing our inflation and momentum terms
- Rest the significance of added variables such as e.g., momentum or the added interaction terms
- Check the validity of the models with more/ different performance metrics
- More thorough analysis of feature importance and selection by using for example a causality-based features selection. Since classical feature selection algorithms, like the one used by us, only selects features based on their correlation to the class variable, Yu et al. (2020) showed that including causal relationships between the features supports building more interpretable and robust prediction models
- Tuning the models with more values for each parameter at once to get the best parameters
- Try to reduce the overfitting of our best model, the Gradient Boosting Classifier

References

Armina, R., Mohd Zain, A., Ali, N. A., & Sallehuddin, R. (2017). A Review On Missing Value Estimation Using Imputation Algorithm. *Journal of Physics: Conference Series*, *892*, 12004. https://doi.org/10.1088/1742-6596/892/1/012004

Barone, A. (2020). *Stock Trading Strategy & Education: Small Cap*. https://www.investopedia.com/terms/s/small-cap.asp

Gautam, C., & Ravi, V. (2015). Data imputation via evolutionary computation, clustering and a neural network. *Neurocomputing*, *156*(4), 134–142. https://doi.org/10.1016/j.neucom.2014.12.073

Gu, S., Kelly, B., & Xiu, D. (2018). *Empirical Asset Pricing via Machine Learning* (w25398). https://doi.org/10.3386/w25398

John, C., Ekpenyong, E. J., & Nworu, C. C. (2019). Imputation of Missing Values in Economic and Financial Time Series Data Using Five Principal Component Analysis (PCA) Approaches. *Central Bank of Nigeria Journal of Applied Statistics*(Vol. 10 No. 1), 51–73. https://doi.org/10.33429/Cjas.10119.3/6

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, É. (2011). Scikit-Learn: Machine Learning in Python. *Journal of Machine Learning Research*(12), 2825–2830.

Ping, X.-O., Lai, F., & Tseng, Y.-J. (Eds.). (2014). *Evaluation of Imputation Methods for Missing Data and Their Effect on the Reliability of Predictive Models*. IARIA. http://www.thinkmind.org/index.php?view=instance&instance=BIOTECHNO+2014

Raschka, S. *Scikit Learn - Support Vector Machines.* Packt Publishing Ltd. https://scikit-learn.org/stable/modules/svm.html#svm-classification

Schmidt, P., Mandel, J., & Guedj, M. A Comparison of six Methods for Missing Data Imputation. *Journal of Biometrics and Biostatistics*, *6*(224), 1–6.

Scikit Learn - Imputation of Missing Values. https://scikit-learn.org/stable/modules/impute.html

Shukla, A. K., Pippal, S. K., Gupta, S., Ramachandra Reddy, B., & Tripathi, D. (2020). Knowledge discovery in medical and biological datasets by integration of Relief-F and correlation feature selection techniques. *Journal of Intelligent & Fuzzy Systems*, *38*(5), 6637–6648. https://doi.org/10.3233/JIFS-179743

Smith, T. (2020). *Stocks: Value Stock*. https://www.investopedia.com/terms/v/valuestock.asp

Tutorial on 5 Powerful R Packages used for imputing missing values. https://www.analyticsvidhya.com/blog/2016/03/tutorial-powerful-packages-imputing-missing-values/

Yu, K., Guo, X., Liu, L., Li, J., Wang, H., Ling, Z., & Wu, X. (2020). Causality-based Feature Selection. *ACM Computing Surveys*, *53*(5), 1–36. https://doi.org/10.1145/3409382

Zucchi, K. (2021). *Stock Markets: Inflation's Impact on Stock Return*. https://www.investopedia.com/articles/investing/052913/inflations-impact-stock-returns.asp

Appendix

Figure 1: Graph showing the variables with the highest percentage of zero values

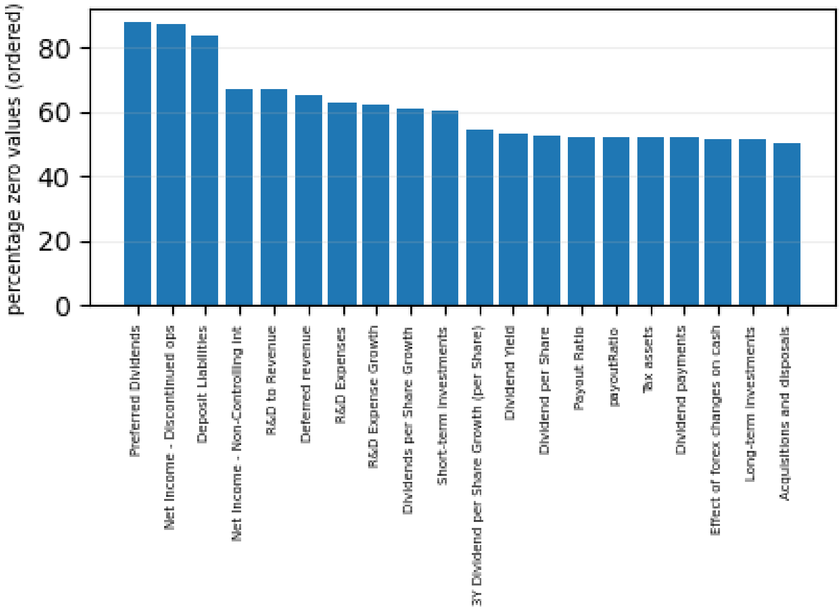
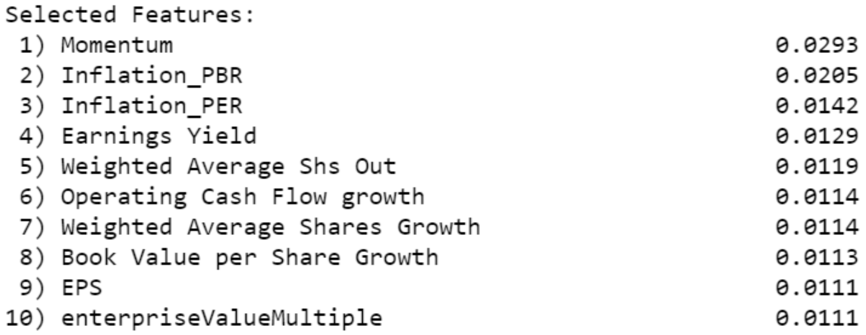


Figure 2: Ten most important features according to Random Forest



```
Selected Features:
 1) Momentum                                         0.0293
 2) Inflation_PBR                                    0.0205
 3) Inflation_PER                                    0.0142
 4) Earnings Yield                                   0.0129
 5) Weighted Average Shs Out                         0.0119
 6) Operating Cash Flow growth                       0.0114
 7) Weighted Average Shares Growth                   0.0114
 8) Book Value per Share Growth                      0.0113
 9) EPS                                              0.0111
10) enterpriseValueMultiple                          0.0111
```

Figure 3: Graph showing all the selected features by the random forest classifier whose importance is higher than the median importance and the cumulative importance.

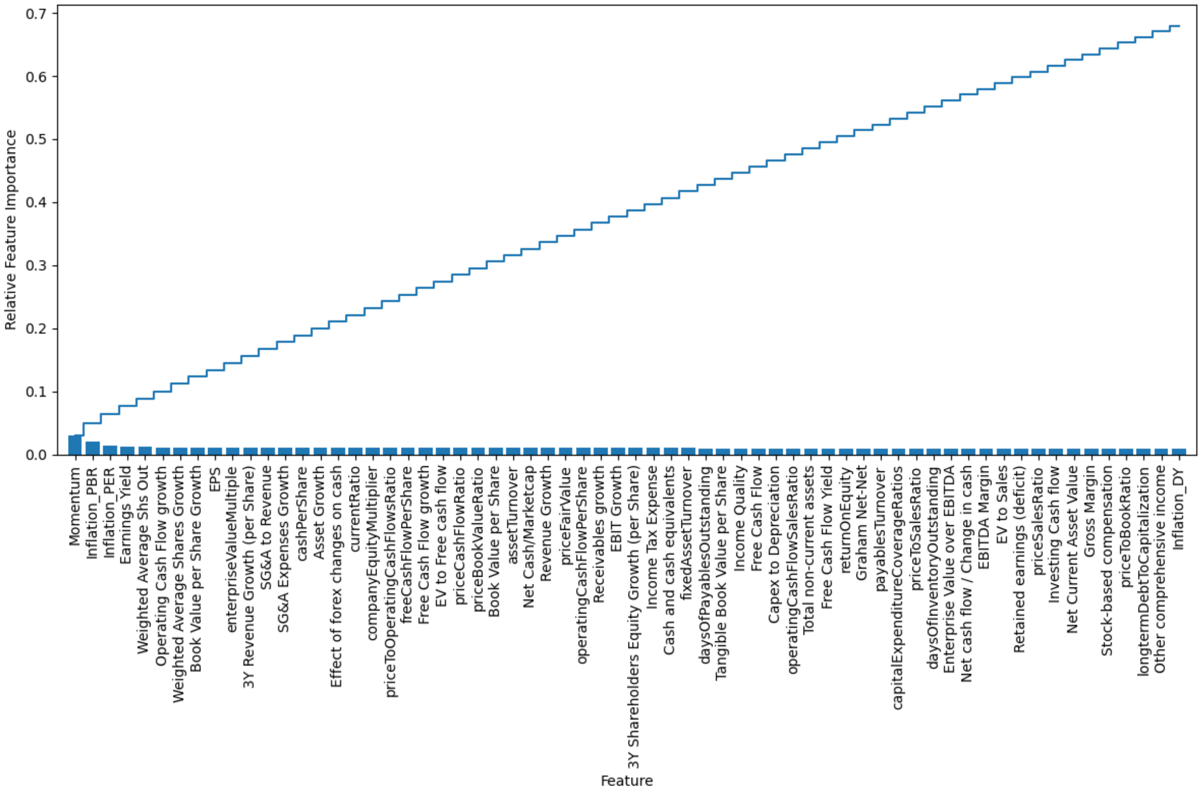


Figure 4: Number of classifications per class before and after up sampling

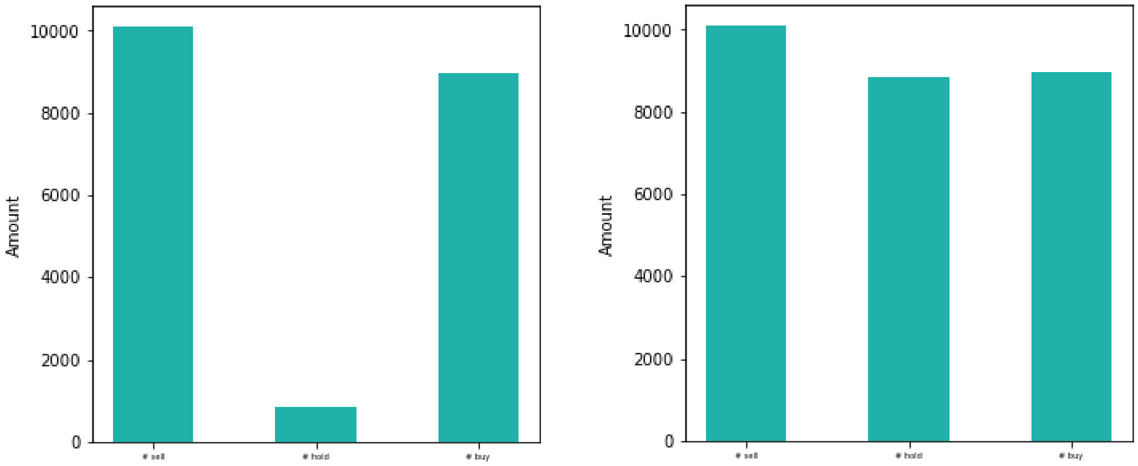


Figure 5: Ranking of all Classifiers by Test Score

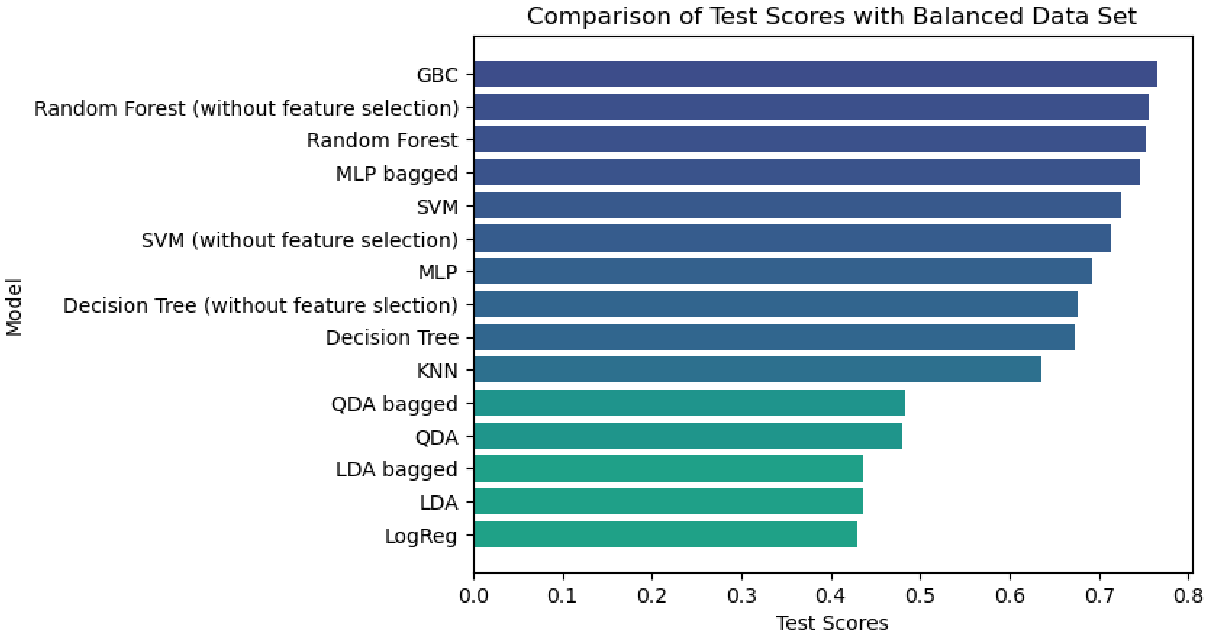
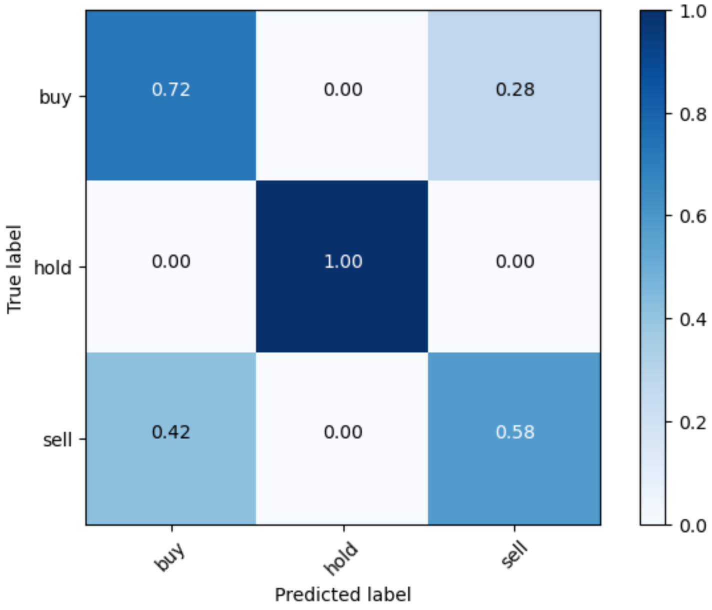


Figure 6: Confusion Matrix of Gradient Boost Classifier


buy
hold
sell
True label
buy
hold
sell
Predicted label