UBS Stock Recommendation Final Report

Jason Wang, Leyi Hua, Ningyuan Du, Yiwen Zhuang, Zhiheng Wang, Zijian Li , Ziyuan Wang

NYU Tandon Department of Finance and Risk Engineering

Author Note

Guidance provided by the Union Bank of Switzerland

UBS Stock Recommendation Final Report

Introduction

Problem Reinstate

In the NYU-UBS summer project, we are the stock recommendation group. We are responsible for recommending holding positions of certain stocks to the portfolio management group. To make the best practice, we apply certain machine learning algorithms to historical data and time series analysis.

Consistent profit from the modern stock market is very difficult due to the randomness of stock price. Any trader or company who wants to beat the market will need a deep understanding of the market behavior patterns. However, with more and more quantitative trading involved in the market today, it is almost impossible for any individual to perform calculations and catch the market patterns manually. Thus, an efficient and more significantly, reliable stock recommendation system is crucial for all interested parties.

Today, most research about stock recommendation system focus on:

1. the recommendation methods based on the stock comment
2. the recommendation methods based on the stock price forecasting

In our project, we hope that we can bring more quantitative measures into consideration regarding the recommendation system. Specifically, we want to figure out which group of stock measures may benefit and provide useful insight when we need to predict the holding position of the stock. We will also try different machine learning algorithms including but not limited to decision trees, dimensionality reduction algorithms, and gradient boosting algorithms lightGBM. We adjust each model and produce the best possible model for prediction.

Ultimately, we want to explore the best model that can output useful insights in different periods. Our model will be based on many quantitative measures of each stock and funds calculated by historical data.

Overview of stock recommender system

Now that the stock recommendation system is a financial model, we must have 4 basic problems or questions for this model.

The first necessary question is: what is the stock recommender system? Stock recommender system, as its name would suggest, is the system for stock recommendation. Recommender systems can catch patterns in stock price movements and develop stock recommendations based on the patterns. Thus, these recommendations can significantly reinforce the decision-making process of a stock trader.

The second meaningful question is: why do we need the stock recommender system? Nowadays, in the stock market, we have an overwhelming number of choices that need to be decided. With a stock recommender system, we can clean, arrange, and efficiently convey relevant information in order to mitigate the problem of information overload, which has created a potential problem for many stock investors. Recommender systems solve this problem by dynamically searching through massive stock information to provide investors with personalized content and services. In addition, it supports stock market traders, individual investors, and fund managers in their decisions by suggesting investment in a group of equities when there exists a great opportunity to make profits. In general, we need it to help us make decisions for stock selection. Our team would like to build a useful and reliable stock recommender system for both individual and institutional investors.

The next core question is: how do we build it? In the beginning, we need to collect information. The system collects relevant information of stocks to generate financial information of stocks for the prediction tasks including financial ratios, number of holdings, or stock prices. Then, it passes into the learning phase. It applies a learning algorithm to filter and exploit the stocks' features from the feedback gathered in the information collection phase. The final stage is the prediction/recommendation phase. It recommends or predicts what kind of stocks the investor may prefer.

The last important problem is: how do we evaluate it? The quality of a recommendation algorithm can be evaluated using different types of measurement which can be accuracy or coverage. The type of metrics used depends on the type of filtering technique. Accuracy is the fraction of correct recommendations out of total possible recommendations while coverage measures the fraction of objects in the search space the system can provide recommendations for. Statistical accuracy metrics include Mean Absolute Error (MAE), Root Mean Square Error (RMSE), and Correlation. Decision support accuracy metrics contain Reversal rate, Weighted errors, Receiver Operating Characteristics (ROC) and Precision-Recall Curve (PRC), Precision, Recall, and F-measure.

Our implementation

The main goal and corresponding contribution to this project consist of several parts. First, our team executed the original python file provided a few times, debugged, and confirmed that we can replicate the result on a test database. Second, we learned to publish the desired tables, metrics, and graphs through an online platform, Weights and biases, by adding compatible codes in the project. Most importantly, we analyzed the result in multiple ways for the purpose of improving upon the current model. The current model took in a time series of investment products a fund could invest in, and output three different lists of stocks. For example, the model is capable of evaluating a list of stocks that Fund ABC was investing in and wasn't investing in January and recommend to the fund manager three different lists of stocks that Fund ABC should hold, buy, and sell in the next month, which is February. However, the limit of the current model is that it does not emphasize the changes in time of the preference of the fund manager. We are to design a more timely-preference-sensitive model, which will take actions such as, Fund buying more aggressive growth stock, into consideration. Our approach is to first analyze the difference of some key factors in time including the consistency of accuracy scores, the difference of importance of variables, predicted probability, and new holdings. Our next step is to enhance the result through feature engineering, and model optimization based on our analysis.

EDA Results and Conclusions

Model EDA

Imbalanced data.

Our data set suffer from extreme data imbalances, specifically, we have more than 98% of data describing the features for is_holding=0 (which means the fund is in a not-holding position for that specific stock at that time), while is_holding=1 only counts a little more than 1%. The imbalance is shown in the below graph.

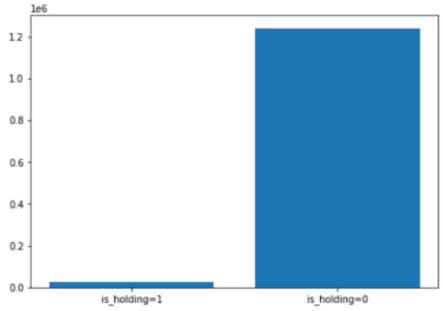


Figure 1. Data imbalance.

Such an imbalanced dataset may cause several serious problems. For example, performing machine learning on an imbalanced dataset may result in a poor performance measure by f1 score even if the accuracy score is high because the model performs very well in the majority class(is_holding=0) while performing very bad in minor class(is holding=1)

There are two possible solutions for imbalanced data (up-sampling and down-sampling). Up-sampling is a mechanism that uses algorithm-generated data points similar to minority class and put into the dataset. Down-sampling is a mechanism that uses algorithms to randomly reduce the training samples in the majority class.

Since each of our datasets is not big enough to suffer the information loss due to the down-sampling, we apply the up-sampling algorithm SMOTE to our data set to create balanced datasets for training. The below graphs show generally how smote works. The blue points are the majority data points and the orange points are the minority data points. On the middle graph, we can see that some new data points(green) are computed and added to the whole data set.

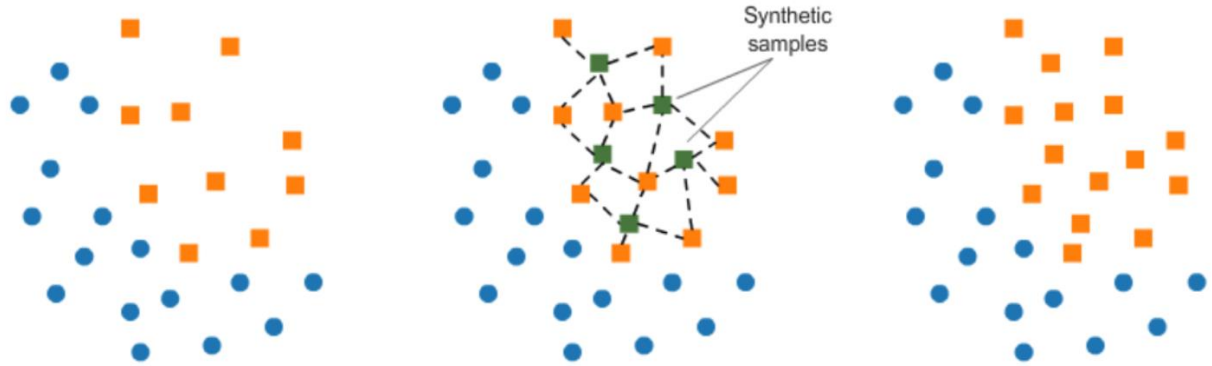


Figure 2. SMOTE example (Analytics Vidhya, 2020).

Accuracy score.

Accuracy score is used as a metric to improve our models. We analyzed the accuracy score metrics differently for our 2 branches of product. First, to recommend an action, there exists a correct answer, which is the next month's data. Thus, accuracy scores can be calculated, which we will explain the importance in a later section. Second, to recommend a similar stock, there is no correct answer, thus we cannot measure our model based on accuracy score. However, we can provide a similarity score.

We have drawn an accuracy score graph for each fund. On average, the accuracy score is 0.91, with a standard deviation of 0.01 and ranges between 0.69 and 1. Here are two examples of accuracy reports for fund 04BBSH-E and 04BG3Y-E.

Figure 3. Accuracy score EDA report of 04BBSH-E and 04BG3Y-E.

Feature importance.

Feature importance refers to techniques that assign a score to input features based on how useful they are at predicting a target variable (Brownlee, 2020). Feature importance is automatically generated by the model which describes the importance score of each feature that was passed into it. The higher the importance score is, the more important the feature is. We have also drawn feature importance stacked line charts for each fund. In summary, we can see that in most of the funds, the most important feature stays the same through time. Here are two examples of feature importance report for fund 04BBSH-E and 04BG3Y-E.

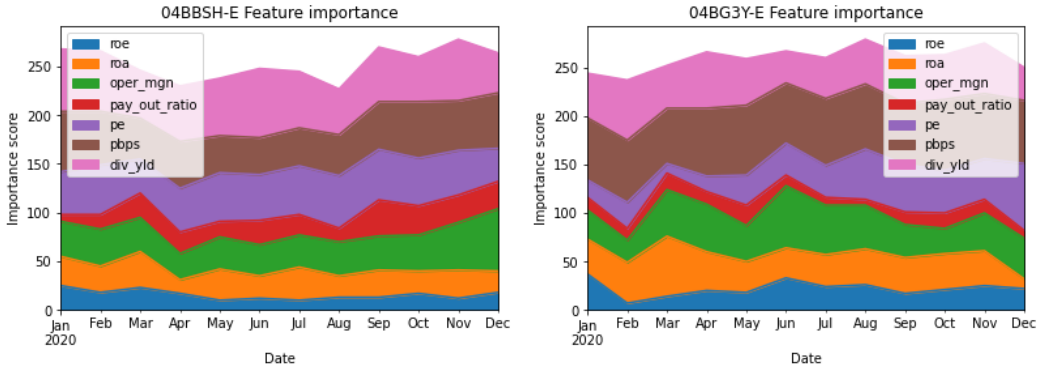


Figure 4. Feature importance EDA report of 04BBSH-E and 04BG3Y-E.

Predicted probability.

Predicted probability is calculated by the built-in function form light_GBM model: mod_lgbm.predict_proba. For example, predicted probability=0.7 means there is a 70% chance estimated by our model that the position for this stock will be "Holding." Thus, the more the predicted probability diverse from 0.5, the more confident our model will be.

We also notice that in most cases for the predicted probability for holdings, the median is around 65%. This is relatively low considering the min is 50% for a holding position. Also, each fund has a different predicted probability, which indicates that our model may perform differently for each model. The below graphs show two predicted probability distributions for two different funds.

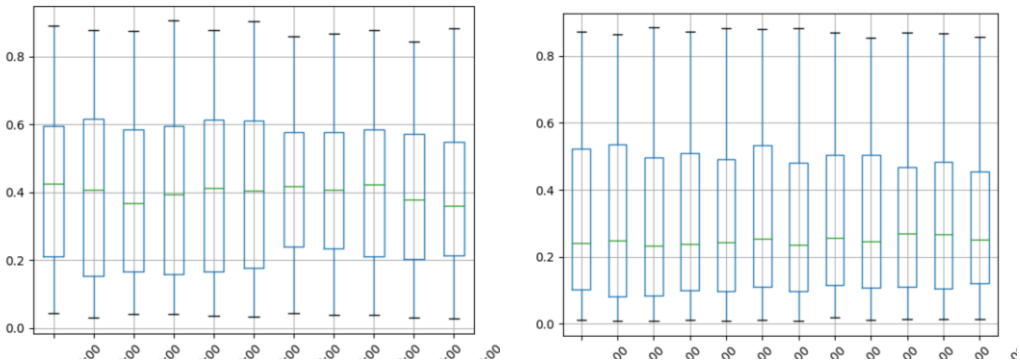


Figure 5. Predicted probability EDA report of 04BBSH-E and 04BG3Y-E.

Shap values.

Shap value is used to explain individual predictions of our default model LightGBM. By allowing us to decompose any prediction into the sum of effects of each feature value, the shap value of each feature shows how much a given feature changed our prediction. The sum of each feature's Shap value plus the base probability would give us the target prediction. For our project, if the target prediction is over 0.5, we would predict holding the stock (is_holding=1).

For each fund, we looped over each month to find the Shap Values of the top 5 holdings. This step enabled us to discover the trend of investors' preferences over time. Here we take the fund Buffalo Small Cap Fund (id 04BG3Y-E) as an example. From the graph below, we could tell that pe's contribution to our prediction raises over time, which implies a stock with high pe might be preferred by the investor of Buffalo Small Cap Fund.

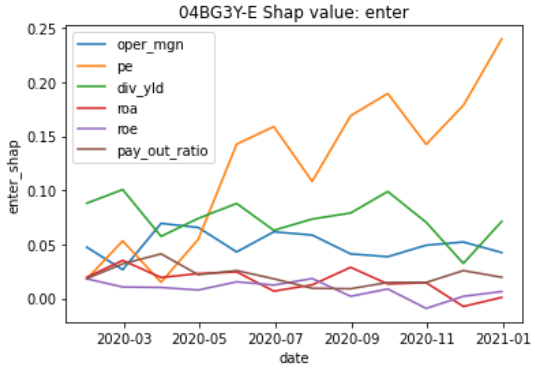


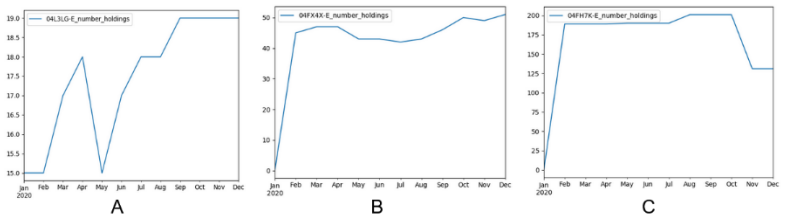
Figure 6. Shap value EDA report of 04BG3Y-E.

Holdings EDA

Number of holdings.

The number of holdings is the sum of all the unique holdings of a fund, equity, fixed income, derivative, or alternative investments included.

The role of the number of holdings is to contribute to diversification. Diversification is a risk management strategy that mixes a wide variety of investments within a portfolio. A portfolio constructed of different kinds of assets will, on average, yield higher long-term returns and lower the risk of any individual holding or security. A well-diversified portfolio contains a mix of distinct asset types and investment vehicles.



A B C

Number	small	medium	large
Long-term Risk	high	low	Very low
Long-term Return	low	high	medium

Figure 7. Number of holdings: long-term risks and return.

In our 3 cases, these 3 funds have small, medium, and large numbers of holdings during 2020. Each of them has different long-term expected returns. The graphs show the number of holdings changes over time.

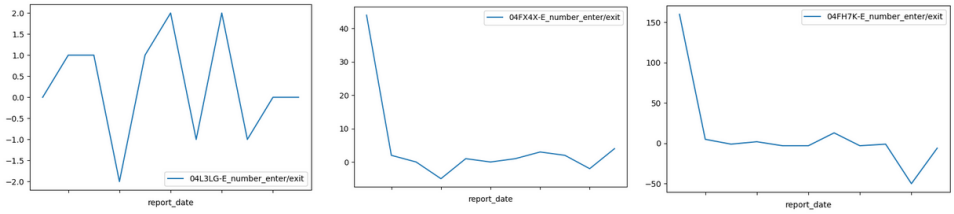
For fund A, a small number of holdings means high long-term risk because the risk diversification is not enough. This also results in a lower expected return in the future.

For fund C, a large number of holdings adequately diversify the risks and promise the return. However, with the very large number of holdings, the fund is like the stock indices such as S&P 500. Therefore, the expected return of this fund is similar to the stock indices and its medium.

For fund B, a medium number of holdings adequately diversify the risks and generate relatively higher returns compared to fund A and C.

Number of positions new/exited.

Number of positions enter/exit refers to the number of new holdings and number of canceled holdings. It reflects the short-term risk of the stock expected by investors and institutions because the short-term fluctuation of the number of holdings affects the confidence and expectations of investors.



A B C

Number	small	medium	large
Short-term Risk	low	medium	high

Figure 8. Number of positions new/exited: short-term risks.

For example, we still have 3 cases above. These graphs indicate the number of positions that enter/exit change over time. The positive number means the number of positions enters and the negative number is the number of positions exits.

Fund A has a small number of positions enter/exit and this means this fund has a relatively stable number of holdings and suffers low short-term risk. Fund B has a medium

volume of the number of positions enter/exit which implies medium short-term risk. Fund C has a high short-term risk due to a large number of positions enter/exit.

Features boxplot over time.

We can get insight on how the fund manager's preference of stocks by analyzing the features of recommended enter, exit, and hold lists. First, we select one feature that we want to analyze. After building the model, we trace back the three different lists of stocks (enter, exit, holding). We then collect the 5 important numbers for the box plot (min, 25 percentile, medium, 75 percentile, max).

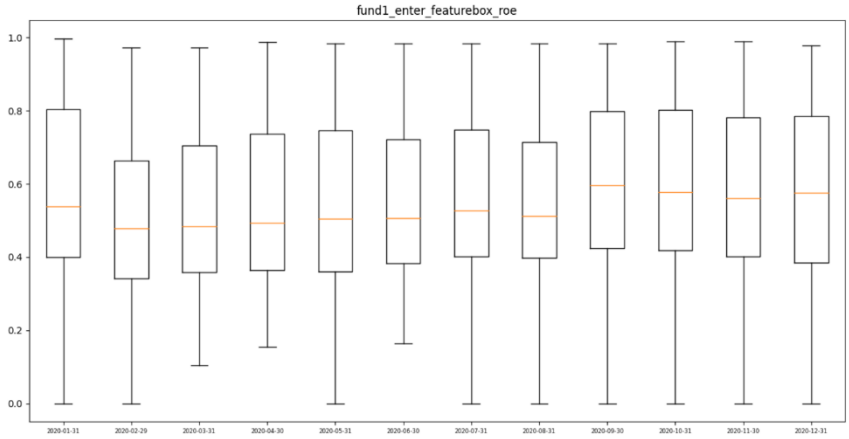


Figure 9. Features boxplot EDA result example.

Here is an example. This is the analysis of the trend of roe of the recommended entering list of one previous fund. By analyzing the trends of the features repeatedly, we can gain insights into the change of preference in stocks of the fund managers.

Enter and exited stock median.

The median of features of new and exited holdings is used to show the trend of the preference of each fund. By calculating the median of each feature of new and exited stocks, we can visualize the preference of the fund for stocks. The result could be used as a reference when we generate predicted recommendations since the consistency between the actual median and the prediction median of each feature could be used to evaluate the performance of the model.

For each fund, we use the holdings of two consecutive months to obtain the new and exited stocks. Select stocks that are held in the first month but not held in the second month as exited holdings. Then new holdings are selected with the same method.

Take '04BBSH-E' as an example, the area plot of the median of features shows us how the preference of the fund changes over time. The graph below shows the above example.

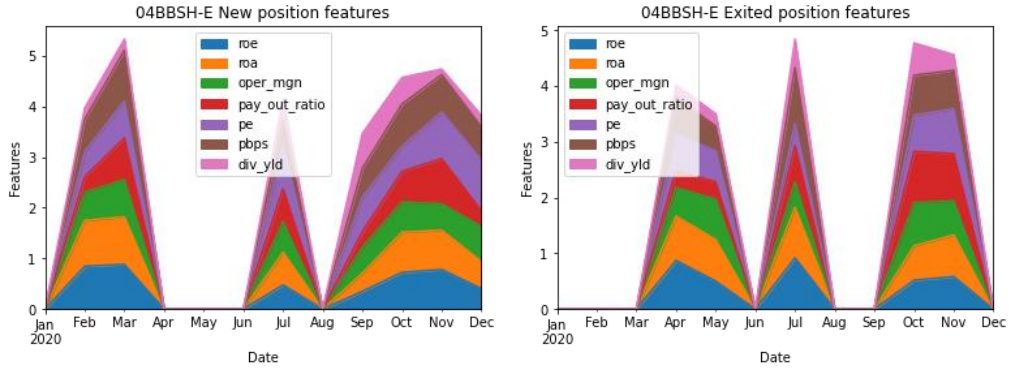


Figure 10. Enter and exited stock median EDA result of 04BBSH-E.

Different Models

Prediction models

The previous EDA indicated that the lightGBM model performs slightly differently in each fund. There are other similar algorithms available. This raises the following six questions: Is the lightGBM model the best possible model with the highest accuracy score for each fund dataset all the time? Which model performs most stabilized? Which model to use for the highest expected accuracy score? Which model is most time-efficient? Which is the best model if we take both the time efficiency and accuracy into consideration? Will our model keep performing well in the future? If not, how to determine when to perform maintenance?

We developed an automatic and user-friendly program to solve these issues by analyzing the performance of each model. The initial build-in models are Logistic Regression, GBM, XGBoots, CatBoost, and Light GBM.

We start by exploring different models. The structure and nature of our recommendation system are suitable for tree models. However, we need a baseline model to compare the performance. Thus, we include the Logistic Regression model in this section as a base model. Any model that performs worse than the Logistic Regression model is considered to fail.

We also include other decision tree models: XGBoost, Catboost. XGBoost, the "GBM killer", was initially a research project began by Tianqi Chen and became famous in 2016. it uses a Histogram-based algorithm for computing the best split. It usually gets very good performance but takes longer than lightGBM.

The key difference between lightGBM and XGBoost is that lightGBM grows trees leaf-wise while XGBoost and other GBMs grow trees level-wise. Most times the leaf-wise algorithm (lightGBM) can achieve lower loss and save times at the same time.

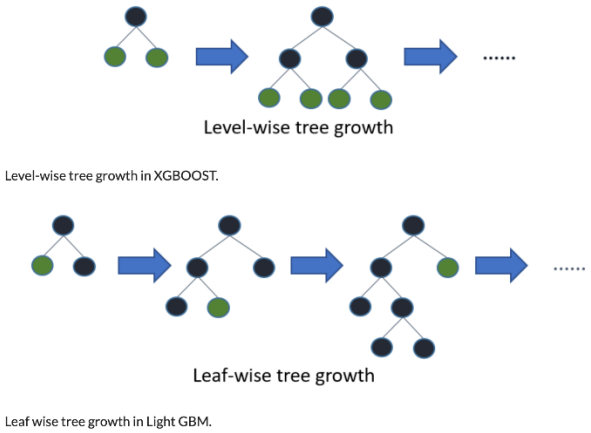


Figure 11. Tree growth comparison between XGBOOST and Light GBM (Swalin, 2018).

While XGBoost does not support categorical data, CatBoost and LightGBM do. For future improvement, support for categorical data is a nice feature to have.

The above algorithms are pre-loaded in the diagnosis programs that we developed, but more importantly, the user may add and edit the algorithm based on their need.

The diagnosis and model suggestion program is meant to be a compliment to our recommender systems. While users can use our stock recommendation system to get suggestions on stock to enter or exit, running the diagnosis program can make sure the stock recommender system functions consistently. The diagnosis program has a menu interface with 6 options shown in the below picture.

```
Menu:
option 1: algorithm with highest acc scores
option 2: algorithm with lowest training time
option 3: best recommend algorithm by customized acc matrix
option 4: most stable model
option 5: time consumption for each model
option 6: show boxplut for accuracy score statistics
option 0: exit model
```

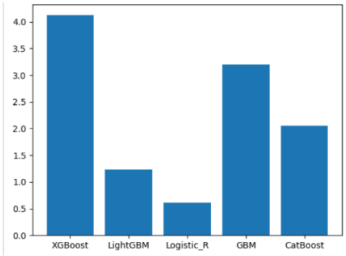
Figure 12. Menu of the diagnosis program.

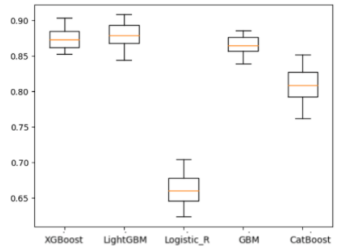
After the user runs the diagnose program and chooses from the menu, the program will perform analysis and provide feedback. The user can verify the performance of the recommending system from option 1 and option 3, other analysis and detailed statistics are also available for users to view from options 2,4,5, and 6. Below is an example of the usage of the program.

```
Your choice: 1
the model with highest score is  LightGBM with avrage score= 0.8775029674044243
the model with second highest score is  XGBoost with avrage score= 0.8759567606843247

Your choice: 2
the fastest model is  Logistic_R with training time= 0.6107547283172607
the second fast model is  LightGBM with training time= 1.2356481552124023

Your choice: 3
How important is the training time to you?
you can also edit the score matric in the source code
please enter a number from 0-10 where 10 is most important and 0 is not important at all
10
the fastest model is  LightGBM with training time= 0.835183836515753
the second fast model is  Logistic_R with training time= 0.7621864410568138
```






```
Your choice: 4
the most stable model is  GBM with variance of scores= 0.00021797120698909876
the second stable model is  XGBoost with variance of scores= 0.00021951704979166213

Your choice: 5
The 1 th model is Logistic_R model, time consumption is 0.6107547283172607 seconds.
The 2 th model is LightGBM model, time consumption is 1.2356481552124023 seconds.
The 3 th model is CatBoost model, time consumption is 2.051792860031128 seconds.
The 4 th model is GBM model, time consumption is 3.195493221282959 seconds.
The 5 th model is XGBoost model, time consumption is 4.123475551605225 seconds.

Your choice: 6
The 1 th model is LightGBM model, score is 0.8775029674044243
The 2 th model is XGBoost model, score is 0.8759567606843247
The 3 th model is GBM model, score is 0.8651356035223613
The 4 th model is CatBoost model, score is 0.810685685423258
The 5 th model is Logistic_R model, score is 0.6635744756461905
```

Figure 13. Demonstration of the diagnosis program.

As long as the LightGBM model performs well in the diagnosis program, our recommended system is considered a good performance.

By applying the diagnosis program to the current datasets, we discovered that in most cases, lightGBM outperforms other models considering the accuracy and time consumption. XGBoost and GBM show similar accuracy scores while taking significantly longer. All gradient Boost algorithms outperform our base model: Logistic Regression. Catboost is the second fast gradient boosting algorithm, but the scores are also lower than XGBM and GBM.

Thus, we focus on analysis and tuning the lightGBM model as it performs well in most cases. In future usage, if the diagnosis program suggests that LightGBM performs badly, then out stock recommend the system is subject to maintains. The user should run the diagnosis program from time to time.

Similarity models

K-means.

Diversification is a pivotal step for constructing portfolios. One effective strategy which enables us to pick stocks that behave differently and help us to mitigate the risk is the K-means algorithm. First, we determined the value “K”, which represents the number of clusters. By applying the Elbow method, we found the most suitable cluster number is 10. We used k-

means++ provided by the Scikit-Learn package to initialize 10 numbers of distinct centroids (Asad, 2020).

Second, we measured the Euclidean distance between each point and the centroid, and assigned each point to the nearest cluster (R, 2020). Third, we reassigned the centroid value to be the calculated mean value for each cluster and reassigned data points to the nearest centroid.

After repeating until data points stay in the same cluster, we got the cluster number of each stock in our data set so that we could differentiate between them while constructing a diversified portfolio.

Though K-means is simple to implement and scale to large datasets, we found it is limited when the number of dimensions increases. For improvement, we would try "spectral clustering" to modify the clustering algorithm.

KNN.

We tried K nearest neighbors (KNN), a great go-to model for recommender system development, to implement the item-based collaborative filtering, which used the actions of users to recommend other stocks. It does not assume the underlying data distribution pattern but uses feature similarity to predict the cluster that the new point will fall into, thus enabling us to recommend similar stocks (D, 2019).

We applied KNN to build a recommendation function that enables users to import the dataset, choose one specific stock, and select some features that they intended to (If the user doesn't select some specific features, the function would recommend based on all features in our dataset), then the system will return stocks that are most like the chosen stock based on the selected features. In addition, the function also allows users to set the number of recommended stocks. The picture below is a brief example of the function input and output (Liao, 2018).

```
stock_recommend(df,'T1N9J9-S-RPMZQ3-R-04BBSH-E', 1)

Most similar stock to stock: T1N9J9-S-RPMZQ3-R-04BBSH-E is: ['WW3RWY-S-Q1LLR2-R-04BBSH-E']

stock_recommend(df,'T1N9J9-S-RPMZQ3-R-04BBSH-E', 3)

Most similar stock to stock: T1N9J9-S-RPMZQ3-R-04BBSH-E is: ['WW3RWY-S-Q1LLR2-R-04BBSH-E', 'BW90JZ-S-SC41Y3-R-04BBSH-
E', 'FBBGRD-S-Q8DBNV-R-04BBSH-E']
```

Figure 14. Demonstration of KNN.

Later, we plan to improve the recommender function by importing data from investing websites to include more stocks, and automatically discerning the user's feature preference to make the function more user-friendly.

LSTM. Given the fact that the dataset could be regarded as time series, we could try algorithms that could consider time dependency. LSTM (Long Short-Term Memory) is an artificial recurrent neural network, which has a unique architecture that enables the model to persist data. The reason for choosing LSTM to compare with the original model is that LSTM can remember information for longer periods of time, which is believed to be a quite beneficial feature in our case. Unlike ordinary RNN, LSTM can explore the effect from the past on current preference.

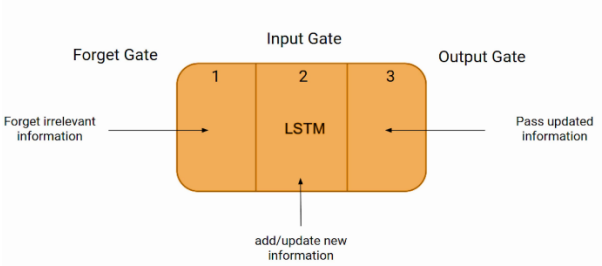


Figure 15. Illustration of LSTM (Saxena, 2021).

For each fund, we compute the median of each feature of all holdings for each month. Then we obtain several time series for all features for each fund. Use multivariate LSTM to predict each feature in the next month with a timestep of 1, that is, we predict the value of each feature based on all past values. We need to fit seven models for each fund to predict the value of each feature. With the most up-to-date predictions, we search for stocks with features that are closest to the predictions as recommendations.

fsym_regional_id	roe	roa	oper_mgn	pay_out_ratio	pe	pbps	div_yld	score
BRVLL3-R	0.608671	0.675781	0.727488	0.386399	0.400824	0.653247	0.311961	[[0.07223930645816383]]
PC3QS5-R	0.814591	0.746659	0.635772	0.386399	0.473108	0.674755	0.311961	[[0.2022245293384194]]
KW9F7H-R	0.812567	0.615028	0.591195	0.386399	0.408176	0.590631	0.311961	[[0.21435569333600454]]
P259NT-R	0.659432	0.729062	0.631294	0.386399	0.596851	0.579400	0.311961	[[0.24753213789680387]]
F2H929-R	0.563856	0.553437	0.621066	0.386399	0.277389	0.528428	0.311961	[[0.25257129340272994]]

Figure 16. Demonstration of LSTM.

From the above result, we see that this algorithm at least works on certain funds with enough data points. However, it does not always work well on all funds. Though time dependency is considered in LSTM, the drawbacks prevent us from exploring further. The dataset becomes quite small for a neural network as we only have one median for each month, which means only twelve data points for one year. Also, the high time cost is the main disadvantage as neural networks take time to train the model. Therefore, LightGBM performs better than LSTM in our case.

LightFM.

During our research of clustering models, we have also tried the LightFM model to recommend similar stocks within the fund. LightFM is a collection of popular recommendation algorithms for both implicit and explicit feedback. LightFM utilizes a hybrid matrix factorization model representing users and items as linear combinations of their content features' latent factors. The matrix factorization model in LightFM is well used in user-item problems, where the interaction matrix will be decomposed into two matrices, the user matrix, and the item matrix. The illustration of these two matrices is shown in the below figure.

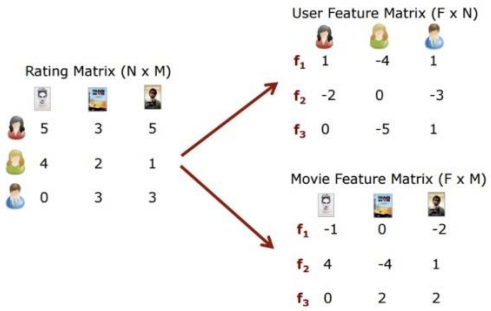


Figure 17. Illustration of Matrix Factorization (MSiA, 2019).

The metrics of light FM are as follows. In this situation, we treat our funds as the users in light FM, and stocks as the items in lightFM. First, each user's value in the user matrix is q_u. Then, each item's value in the item matrix is p_i. Then light FM predicts by calculating a dot product of user and item matrix with bias $\widehat{r_{ui}} = f(q_u \cdot p_i + b_u + b_i)$ (Ullah, 2019). The factors used can be also explained in the below manner.

The latent representation of user u is given by the sum of its features' latent vectors:

$$\boldsymbol{q}_u = \sum_{j \in f_u} \boldsymbol{e}_j^U$$

The same holds for item i:

$$\boldsymbol{p}_i = \sum_{j \in f_i} \boldsymbol{e}_j^I$$

Figure 18. The formula for user and item latent vectors (MSiA, 2019).

The advantage of using LightFM is that its performance exceeds pure content-based models when the user's information is considered in the data. Also, it is suitable for abundant data like the current dataset we have. However, LightFM is unable to incorporate the time series into the model, thus, lightGBM is better to use in this case.

Problems Resolved

Introducing time-varying preferences to the model Prediction models

Method 1: Seasonal preference.

The previous model provided by the manager consists of a structure of training and predicting one same month at a time. The performance of such a model is not ideal according to the f1 score records of models predicting different months. The prediction accuracy on currently held stocks is extremely low which is around 15%. The application is also constrained by the structure of this model. The model strictly requires certain amounts of the stock deals from the current month to train and produce recommendations. It may suffer from insufficient data and short response time.

We believe that models ought to predict future tradings concerning recent strategies. To achieve this goal, the model is supposed to encode recent purchasing history and analyze the strategies implied. As a result, we believe that it is necessary to build a model that takes time-varying preferences into account.

To achieve the goal mentioned above, the model needs to be capable of encoding not only data from the current month but also the data from previous periods. We, therefore, decide to design a model that is trained on data from three previous months and predicts results for the

current month. The newly designed model is constructed based on the light GBM tree structure, with a maximum depth of 8 and several leaves of 40. We believe that with this structure, the model is capable of learning the trading strategies of the funds from recent periods since the data from the recent three months could provide enough information. We also believe that such a structure can mandatorily make the model get a bigger picture on the trading strategies since broader data was given and compared to the previous model setting, the effect of randomness could be reduced. Hence the accuracy of the prediction could be improved.

After training the data, the accuracy can reach over 96%. Compared to the previous model, the new data structure is more comprehensive and inclusive. Fitting the data of continuous three months to predict the following one month could efficiently prevent us from suffering the short response time.

To further investigate the model accuracy, we conducted an ablation study on model parameters. The model was trained and tested multiple times on the same group of training data and testing data for different parameter settings. The results, as shown in the table below, reveal that maximum depth possesses the greatest influence on accuracy. As the maximum depth increases, the accuracy increases proportionally. The number of leaves, on the contrary, does not have a significant effect on the accuracy. When a maximum depth is high, the accuracy has only a subtle increase as the number of leaves increases. The effect, however, basically vanishes after the number of leaves increases beyond 30.

Maximum depth/number of leaves	15	20	30	40	50
3	0.815	0.815	0.815	0.815	0.815
4	0.872	0.867	0.867	0.867	0.867
5	0.898	0.905	0.905	0.905	0.905
6	0.924	0.925	0.938	0.941	0.942
7	0.933	0.947	0.955	0.960	0.958
8	0.937	0.953	0.965	0.966	0.964

Table 1. Ablation study for model parameters.

Certainly, the newly designed model is not flawless, there are a few disadvantages, for instance, though the new model was trained with broader data from the previous three months, the attention is distributed evenly to all three months (by convention, more recent data should be more attention focused). There is no process to lift the weight of attention to more recent data. It may result in a drop in accuracy when a fund changes investment strategies rapidly in one month.

Method 2: New changes as a weight on data.

For this method, we have transformed our compromised idea into something useful in our model. Our initial idea is that we will use the last month's condition as a feature in the training of this month's model. The advantage of this method is that it directly links the month's fund preference to this month's data. However, the disadvantage of this method is that new changes are highly correlated to the label of this month, thus making it unreasonable to be treated as a feature. Then, we thought of transforming this idea into weight more on data entries that have a new change. We first designed this new feature named new_changes, which follows the logic below.

Action	Last month's holding	Current month's holding	New changes
Stock entered	0	1	1
Stock considered	Non-existent	1	1
Stock exit	1	0	1
No action			0

Table 2. New changes illustration.

When last month's holding is 0, and the current month's holding is 1, new changes are 1, which representing a stock that has entered. Similarly, we made four cases of actions, they are, stock entered, stock considered, stock exit, and no action. After we have obtained the values of new changes which treat the action with equal importance, we can proceed to weigh more on them using the following two methods we are going to introduce. In this method, new_changes treats the three conditions of new changes in stock holding compared to last month equally.

Method 3: Increasing sample size by duplicate.

Since we have found a way to label the newly changed data from the other, one promising idea is to just weigh more on those data to capture the preference change in time. We modified the training dataset by increasing the "newly_changed" data to a certain percentage, build the model, and then test it on the original dataset. For example, there are 100 stocks for Fund A, dated February. We first selected those stocks with new_changes = 1 (just bought, sold, or added to the dataset from January), and simply duplicated them to 20 percent compared to those that are unchanged from January. We then run the model with this modified dataset and test the model on the original, clean February dataset. We then see an obvious increase in the accuracy. Here we attached a graph.

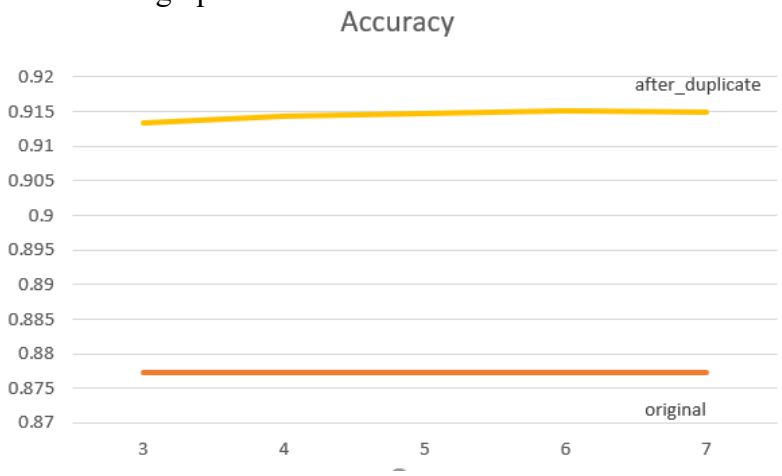


Figure 19. Accuracy score of increasing sample size by duplicating.

Method 4: Increasing sample size by double smote.

As we have mentioned previously, smote is an upsampling method used to increase the amount of certain data. Thus, to combine the goal of increasing the samples which have new_changes = 1, we can first set smote's label to new changes to duplicate the entries that have new_changes=1. Then we can drop new_changes from the data frame and assign the true label as

the smote target to resolve true unbalanced data. The following is a diagram that illustrates this process.

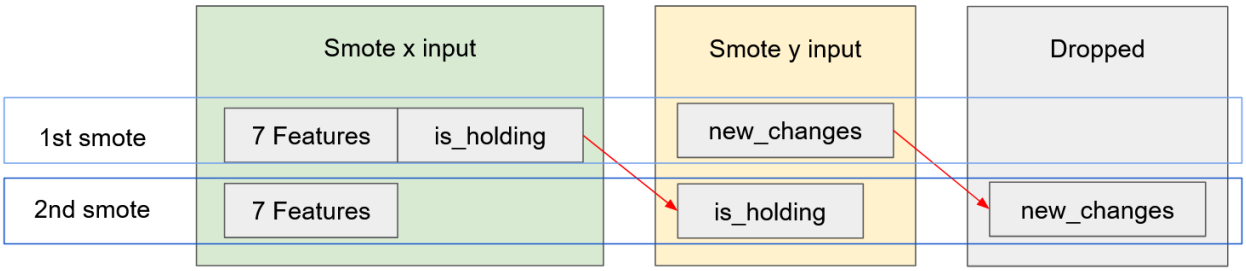


Figure 20. Double SMOTE illustration.

Before the first smote, the smote x input is 7 true features plus the true target, the smote y input is new changes. Between the first and second smote, the true target was moved to be the smote y input, then new changes are completely dropped from the data.

In this way, we have added randomness into the duplicating of the new_changes = 1 samples compared to simply duplicating by concatenating is_holding and new_changes.

Accuracy score metrics

During our search for a proper accuracy score, we have tried multiple methods. The first accuracy score measure we used was the plain F1, which conveys the balance between the precision and the recall. Thus, $F1 = \frac{2*precision*recall}{precision+recall}$.

However, the F1 score only considers the learning rate of 1 and not 0. In our case, the prediction of 1 and 0 are both important. Then, we utilized the weighted-F1 score. According to sklearn, weighted-F1 score calculated metrics for each label, and find their average weighted by support (the number of true instances for each label) (sklearn.metrics.f1_score, 2021). This method is used to account for label imbalance but it can result in an F-score that is not between precision and recall.

At last, following the metrics of weighted F1 score and the screen task, we also developed an accuracy score measurement. Our method is as follows. First, we obtain the is_holding of this month and next month, then we train the model using this month's data. We will be recording an action of change that has the same logic as the screen to calculate the accuracy score. The illustration of the comparison between true action and predicted action is shown below in the table.

True action	This month's holding	Next month's holding
Predicted action	This month's holding	Predicted next month's holding

Table 3. New accuracy metrics explained.

Excessive runtime (multi-threading)

To improve the runtime of calculating accuracy score, F1 score, and F1-weighted score of stock held by each fund on each day, we implement a multithreading method to improve the efficiency of the code. In python and other languages, a thread is an entity within a process that can be scheduled for execution. Also, it is the smallest unit of processing that can be performed in an OS (Operating System).

Generally, the multithreading method utilizes more than one thread to work for one task together. This can dramatically cut down the runtime of the original code. Since we are

processing extremely large datasets, we split them into 8 smaller ones and each one is processed by one thread. As a result, the original runtime, which is over 4000 seconds, has now been reduced to between 600 and 750 seconds. Below is a code snippet of our multi-threading implementation.

```
#Multithreading
if __name__=='__main__':
    start = time.time()
    print('This is the main thread: {}'.format(threading.current_thread().name))
    thread_list = []
    for k in range(1, 9):
        t = threading.Thread(target=accuracy, args=(k,))
        thread_list.append(t)
    for t in thread_list:
        t.start()
    for t in thread_list:
        t.join()
    end = time.time()
    print("Total time is {} seconds".format((end - start)))
```

Figure 21. Code snippet of multi-threading.

Train test split

Through our discussion of discovering a model to evaluate the change of a fund manager's preference by time, adjusting the original code structure and separate train and test dataset is crucial. Upon the original code provided, we modify the structure so we can feed different datasets, (as Train dataset and Test dataset), into the Tasks file to construct our model. By separating the Train and Test dataset, we can modify one specific table for training, such as duplicating/smoting newly changed stocks, or extend the time horizon to include the last three months, build the model, and then test it on the original, clean, one-month table.

Conclusions

We are instructed to comprehend the model that we are provided to recommend stocks based on a fund universe dataset, performing EDA analysis, and modify the current code to incorporate the change of preference in time to create a new model. Here is our conclusion on our project.

Firstly, we have completed a thorough EDA on the dataset and improved our baseline model based on it. We have drawn different conclusions with each of the EDA graph analyses we performed and gained important insights on how the fund and fund managers behave in our given time series.

Secondly, we expanded our selection of models and compared them for best fitting solutions. Our comparison revealed that lightGBM outperformed all the other models in execution time, time-series inclusion, and accuracy.

Thirdly, we have generated a final product that gives recommendations of buy or sell and recommendations of similar stocks in a fund based on user input of fund, stock, and date. This final product concludes all of our previous findings in EDA and model testing and delivers the best performances.

References

Analytics Vidhya. (2020, July 23). *10 Techniques to deal with Imbalanced Classes in Machine Learning.* Retrieved from Analytics Vidhya: https://www.analyticsvidhya.com/blog/2020/07/10-techniques-to-deal-with-class-imbalance-in-machine-learning/

Asad, S. M. (2020, August 18). *AI Movies Recommendation System Based on K-Means Clustering Algorithm.* Retrieved from Medium: https://asdkazmi.medium.com/ai-movies-recommendation-system-with-clustering-based-k-means-algorithm-f04467e02fcd

Brownlee, J. (2020, March 30). *How to Calculate Feature Importance With Python.* Retrieved from Machine Learning Mastery: https://machinelearningmastery.com/calculate-feature-importance-with-python/

D, R. (2019, April 2). *k-Nearest Neighbors.* Retrieved from Medium: https://medium.com/@rndayala/k-nearest-neighbors-a76d0831bab0

Last Name, F. M. (Year). Article Title. *Journal Title*, Pages From - To.

Last Name, F. M. (Year). *Book Title.* City Name: Publisher Name.

Liao, K. (2018, November 10). *Prototyping a Recommender System Step by Step Part 1: KNN Item-Based Collaborative Filtering.* Retrieved from towards data science: https://towardsdatascience.com/prototyping-a-recommender-system-step-by-step-part-1-knn-item-based-collaborative-filtering-637969614ea

MSiA. (2019, April 24). *Personalized Restaurant Recommender System Using A Hybrid Approach*. Retrieved from MSiA Student Research: https://sites.northwestern.edu/msia/2019/04/24/personalized-restaurant-recommender-system-using-hybrid-approach/

R, A. (2020, October 4). *Understanding Clustering in Unsupervised Learning.* Retrieved from Medium: https://arifromadhan19.medium.com/understanding-clustering-in-unsupervised-learning-b0d7a5f61f03

Saxena, S. (2021, March 16). *Introduction to Long Short Term Memory (LSTM)*. Retrieved from Analytics Vidhya: https://www.analyticsvidhya.com/blog/2021/03/introduction-to-long-short-term-memory-lstm/

sklearn.metrics.f1_score. (2021). Retrieved from scikit learn: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html

Swalin, A. (2018, March 13). *CatBoost vs. Light GBM vs. XGBoost.* Retrieved from towards data science: https://towardsdatascience.com/catboost-vs-light-gbm-vs-xgboost-5f93620723db

Ullah, N. (2019). *LightFM Hybrid Recommendation system*. Retrieved from Kaggle: https://www.kaggle.com/niyamatalmass/lightfm-hybrid-recommendation-system