**HKBU - COMP7015 Artificial Intelligence**

**Final Project Report**

**Predicting First Day of IPO Stock Return Using XGBoost**

**Abstract**

Machine Learning has been adopted long in finance and the area of stock trading. Models can be trained to identify technical indicators, optimise portfolio diversification and build automated trading system. First day of IPO stock return is one of the market anomalies being studied by researchers. Pricing of IPO stock is an arena between the management of the listing issuer, stock underwriters which then sell the shares to investors. The outcome of the subscription price is often thought to deviate from the true market value, which can be reflected by the public market on the first day of trading.

Before and shortly after the 21st century, the initial IPO return in the field of artificial intelligence was not studied extensively beyond linear classification. Since then, genetic algorithms to predict IPO underpricing were proposed by Luque, Quintana and Isasi in 2012. In the same year, Huang et al. proposed genetic-search model IPO first day return based on common financial metrics as data inputs. Artificial neural networks (ANN) and support vector machines (SVM) were also proposed later. Quintana, Saez and Isasi proposed an ensemble method, namely random forest, to predict IPO underpricing in 2017, and ensemble method has become a mainstream in studying the topic of IPO returns. In this study, we will explore other evolving and popular artificial intelligence algorithms, namely XGBoost of the ensemble method, to learn IPO data and predict the first day of IPO return. The investable universe of this study is IPO in Hong Kong stock market.

**Introduction**

Extreme Gradient Boosting, often known as “XGBoost”, is an emerging algorithm in machine learning for classification or regression. XGBoost, which is also an ensemble machine learning method, is powerful and efficient that a number of competition winning machine learning algorithms have been based on XGBoost since its emergence. This study is motivated to apply XGBoost to learn first day IPO return for new stocks that were traded on the Hong Kong Stock Exchange (SEHK), with reference to the repository at <https://github.com/crownpku/hk_ipo_prediction> (herein referred to as the “**Crownpku Repo**”). We will also compare XGBoost with another ensemble learning method, namely AdaBoost, which is a fundamental boosting algorithm.

**Input and Output**

A dataset of IPO stocks over the last three years with a number of select attributes is obtained from AAStocks.com, which is a leading Hong Kong financial market real-time data provider. We have tried to reproduce the same attributes as those in the Crownpku Repo, but AAStocks.com had reduced the amount of data to be provided and we could only obtain a subset of the attributes. Table 1 below provides the attributes which we selected from AAStocks.com and employed for our XGBoost model.

|  |  |
| --- | --- |
| **Attribute** | **Description** |
| Ticker | The stock code of the stock |
| List\_date | The listing date of the stock |
| Market\_cap | Range of market capitalization of the stock before IPO |
| Low\_offer\_pr / Upper\_offer\_pr | The lower and upper range of the IPO offer price |
| Final\_offer\_price | The final offer price fixed for the IPO |
| Over\_sub\_ratio | The over subscription ratio, the shares subscribed over the shares issued to the public |
| Allot\_odds | The odds for the investor being allotted for the shares subsubscribed |
| Firstday\_return | The return of the IPO on the first day of trading |

**Table 1**

**Brief Introduction of XGBoost**

Boosting has been an increasing popular ensemble learning methods, whereby multiple weak learners are trained to solve a learning problem and configured to improve the combined prediction results. Weak learners individually are thought to have low variance. They have fast learning speed but due to their simple model configuration. But due to the simplicity, the bias associated with them is high too. Generally, the weak learners are short decision trees with only a few levels known as stumps (also the model choice of XGBoost), and ensemble models configure the weights of each weak learners to maximise the likelihood of the outputs.] Common boosting algorithms are Adaptive Boosting (AdaBoost) and Gradient Boosting, and Tianqi Chen and Carlos Guestrin of the University of Washington modified the Gradient Boosting and coined the term XGBoost as a supervised learning.

The objective function of XGBoost comprises training loss function and regularization penalty for overfitting:

obj(θ) = L(θ) + Ω(θ)

where L is the loss function and Ω is the regularisation term where mean squared error and logistic loss are often chosen as the loss function.

Similar to a general to typical decision tree ensemble model, XGBoost iterates through the building of a classification or regression tree sets where a CART has a score on each leaf which is the prediction score (some refer it as the similarity score). The final score is the sum of the score predicted by each individual tree and the prediction function is thus expressed as (by using forward stagewise algorithm to learn the additive model):

Where *K* is the number of trees (Classification And Regression Trees), *f* is a function (regression tree) in the set of function space.

In training the trees, the key is to defining a suitable objective function and minimize the loss or maximise the likelihood. XGBoost has regularizing parameters to restrict the structure of the trees and configures the prediction scores of the CARTs. An additive strategy is employed, learning one tree at a time and adding a new tree next. Therefore, the prediction model at step t is shown in the formula (by using greedy algorithm to reach local optimization):

For mean squared error to be considered for optimising the objective function, the objective function can be rewritten as (from the derivation of second order of Taylor Formula and removing all constants, specific objective function at time step *t*):

Obj(t)= 

Where hi and gi are defined as:





It is important to regularise the learning objective to prevent overfitting. The complexity is defined by:



In defining Ω(f), i.e. the complexity of the tree, let us first refine the definition of the tree ft(x) as:

 ft(x) = , ,q:{1,2,...,T}.

Without going through too much of the mathematical detail here, we can obtain:

Obj(t)

=

Where i is the i-th datapoint while j is the j-th leaf. We could further simplify the equation above by further defining and :

Obj(t)=

wj are independent to each other above, and then with the quadratic term    and the best wj for q(x), optimal objective function is resulted from:

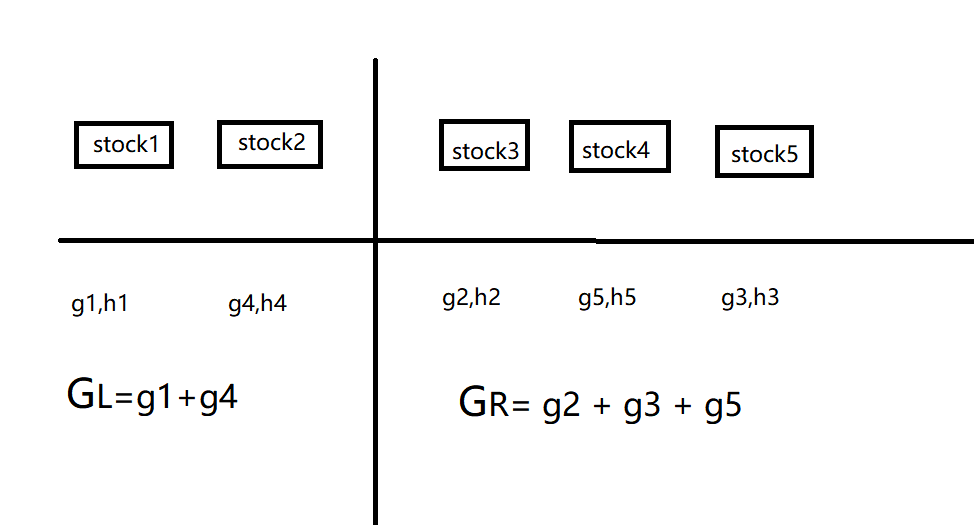




In addition to evaluating how good a tree can be from the above, it would be ideal that we can also enumerate all trees and choose the best one. But it can be done by optimizing one level of a tree at a time and splitting a leaf node into two leaf nodes, and see if there is gaining in score by calculating the gain function:



There is an efficient way to search for an optimal split, where we can sort the instances in order, like the following picture.



**Fig. 1**

The approach above should work well under normal condition, except that certain edge cases which have resulted in a generate model.

**Dataset and Features**

The target investable universe is Hong Kong stock market and our objective is to examining the first day of return of IPO stocks. The dataset is a small dataset obtained from AAStocks.com and contains 520 samples of IPO issuers listed on SEHK between 2018 and 2020. The qualitative and quantitative attributes which we found relevant are summarised in Table 1 above.

Consistent with other studies, first day of return (R) is defined as (closing price of first day of trading – final offer price) / (final offer price), expressed as percentage. The closing price of each stock on the first day of trading is also easily accessible from AAStocks.com.

***Initial dataset***

AAStocks.com provides current and past IPO information up to the last three years. The data is rendered in tabular format and can be extracted by scraping by writing a simple python program. The scraped raw data contains 520 stocks listed from January 2018 to December 2020. The initial dataset is stored in “stock\_file.csv”.

***Pre-processing***

In order for our XGBoost model to learn, the dataset has to be pre-processed, the following summarise some necessary pre-processing steps.

First of all, we analysed the missing values (marked as N/A for a number of fields in the raw dataset) and special values (marked as “認購不足” in the attribute “over subscription ratio”, meaning the IPO was undersubscribed). For those with N/A values, it is noted that AAStocks.com treated the transfer of listing from GEM (historically known as “Growth Enterprise Market”) to the Main Board as IPO and also there were some secondary listings in Hong Kong, both of the cases did not involve public subscription of new shares. We consider it appropriate to drop all such entries with N/A values and dropping such data should not affect the overall integrity of the dataset. For those undersubscribed IPO, though it intuitively suggests that these stocks are less attractive and less in demand and it may be interesting to have a machine learning algorithm to learn the first day return of these undersubscribed stocks, the data from AAStocks.com, unfortunately, did not provide the quantitative information on the extent of the undersubscription. Therefore, for simplicity and demonstration in this project, we also dropped such undersubscribed stocks in the cleaned dataset. After this process, there are 416 samples in the dataset.

Secondly, when the data is scraped, Python reads the data from the csv file as an ‘object’ datatype. Converting an ‘object’ to numeric is necessary for the learning model to read the data. Treatments of string data such as removing ‘,’, ‘%’ and ‘-‘ were performed. The raw data of the market capitalisation attribute was a range of market capitalisations based on the high end and low end of the subscription price. New attributes of “market capitalisation low” and “market capitalisation high” were created to store the low end and high end of the market capitalisation range, respectively and the market capitalisation attribute is disregarded in the subsequent learning.

The final dataset ready for learning is stored in “processed\_stock\_data.csv”. The pre-processing process of the raw dataset is demonstrated in “stock\_data\_cleaning.ipynb”.

**Experiments / Results / Discussion**

***Libraries and parameters***

We use Python as our programming language. To use XGBoost, the library py-xgboost is installed, together with other necessary libraries including numpy, pandas, scikit-learn and matplotlib.

***Hyperparameters of XGBoost***

Documentations of XGBoost set out general parameters and parameters for tree boosters. The following are the important one we modelled:

learning\_rate: step size initially set at 0.1

max\_depth: which govern the training stumps on the depth level they could have and is initially set to 5.

subsample: subsample ratio of the training instances. setting it to 0.5 means that XGBoost would randomly sample half of the training data prior to growing trees. and this will prevent overfitting.

colsample\_bytree: the subsample ratio of columns when constructing each tree, will lead to overfitting if it is set too high. The value is set at 0.3.

n\_estimator: number of training stumps to be built, which is set at 100

objective: the default of objective is mean squared error (regression with squared loss)

In the Jupyter Notebook, a dictionary is defined to store the above hyperparameters:

|  |
| --- |
| params = {“objective”: “reg:linear”, “colsample\_bytree”: 0.3, “learning\_rate”: 0.1, “max\_depth”: 5, “alpha”: 10} |

**Results**

Estimating the first day return of the IPO is a regression problem. We applied the XGBoost library of Python and called the XGBRegressor() class to instantiate an XGBoost regressor object. XGBoost is capable of learning classification problems, which could be done by calling the XGBClassifier() class instead. In Jupyter Notebook, the following lines of code was executed:

|  |
| --- |
| rg\_reg = xgb.XGBRegressor(objective = ‘reg:linear’, colsample\_bytree = 0.3, learning\_rate = 0.1, max\_depth = 5, n\_estimators = 10)  xg\_reg.fit(X\_train, y\_train)  preds = xg\_reg.predict(X\_test) |

Please note that X\_test, X\_train, y\_train are the testing dataset, training dataset and trained results, respectively. Then, root mean squared error (rmse) is calculated by invoking the sklearns’ metrics module. The rmse first obtained from our model was approximately 0.5763.

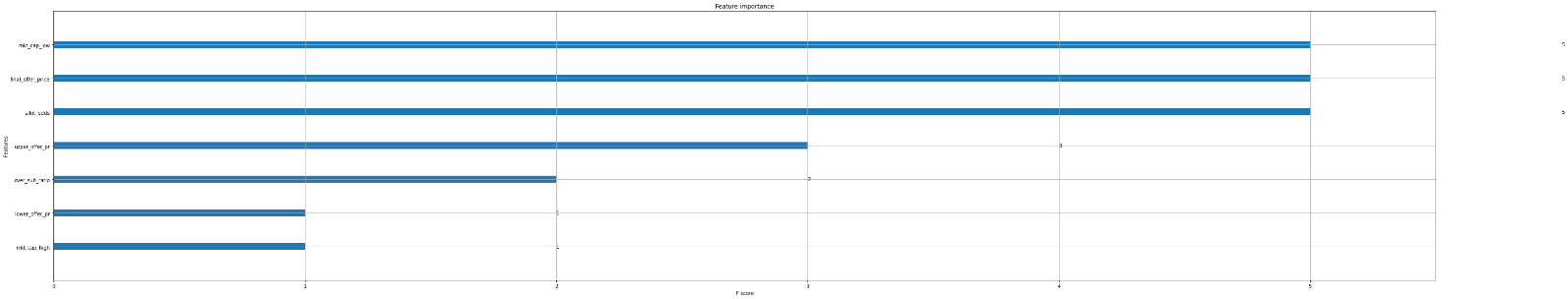
A general practice for improving the training model is to perform k-fold cross validation, where all the entries in the original dataset are used for both training as well as validation, and all entries are used just once. K-fold cross validation is supported by Python’s xgboost using a cv() method. To do this for our learning model, the following line of code was run to build a 3-fold cross validation model and store the results in cv\_results:

|  |
| --- |
| cv\_results = xgb.cv(dtrain=data\_dmatrix, params=params, nfold=3, num\_boost\_round=50, early\_stopping\_rounds=10, metrics=”rmse”, as\_pandas=True, seed=123) |

Please note that data\_dmatrix was defined in our Jupyter Notebook as the Dmatrix data structure for storing our dataset, which is a feature of the Python’s xgboost library.

The 3-fold cross validation successfully achieved improvement in rmse, which now decreased to approximately 0.4694.

Visualisation is made possible by XGBoost, and one way is to examine the importance of each feature column in the original dataset within the model. There is a simple way of counting the number of times each feature is split across the boosting rounds, with the features ordered according to the number of times the features appeared. This can be done by calling the plot\_importance() function in Python’s xgboost library. The resulting chart (Fig. 2) in our Jupyter Notebook is shown below:



**Fig. 2**

From the chart above, it is noted that the lower end of the market capitalisation, final offer price and the allotment odds are the features of importance. However, the final offer price and the allotment odds are normally not available at the time of subscription.

Results of the above are in the Jupyter Notebook “xgb-learner1.ipynb”.

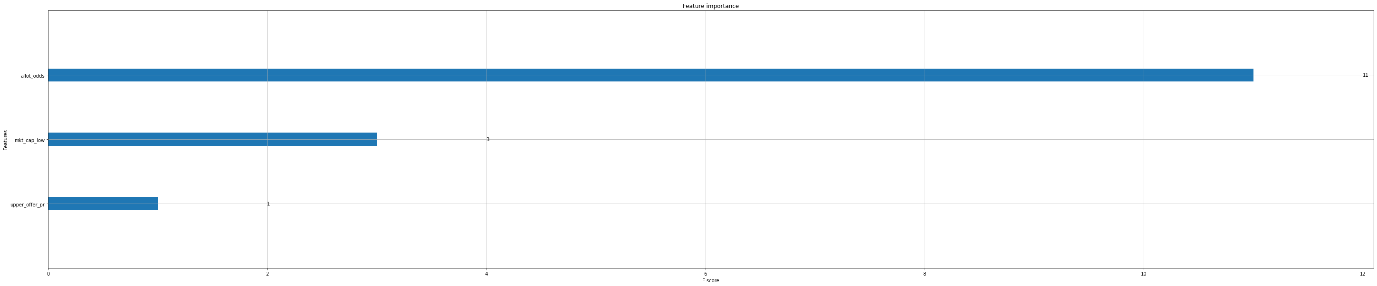
***Changing Hyperparameters***

We then reperformed the machine learning algorithm by changing the params defined above to below:

|  |
| --- |
| params = {“objective”: “reg:linear”, “colsample\_bytree”: 0.2, “learning\_rate”: 0.5, “max\_depth”: 3, “alpha”: 10} |

The changes involved modifying “colsample\_bytree” to 0.2 and “max\_depth” to 3, and more notably the “learning\_rate” is an exaggerated 0.5.

The resulting rmse was slightly increased, to 0.4786. In the previous setting, lower end of the market capitalization, final offer and the allotment odds were the important features. But with these modified hyperparameters, allotment odds becomes the single important feature. The chart extracted from Jupyter Notebook is shown below (Fig. 3):



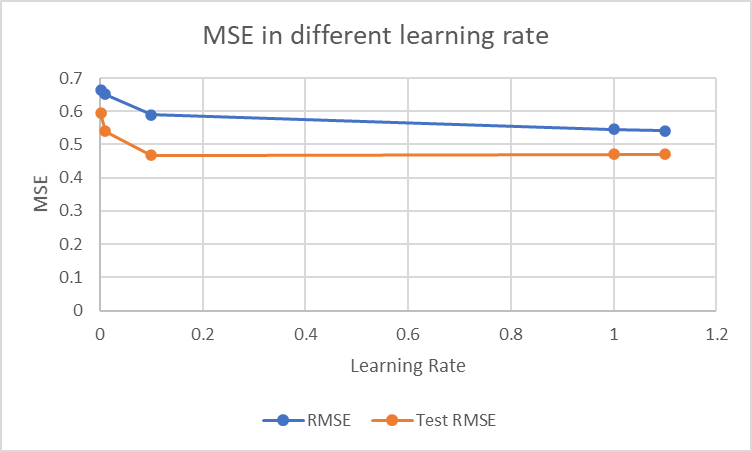
**Fig. 3**

Results of the refined XGBoost training are saved in the Jupyter Notebook “xgb-learner2.ipynb”.

**Comparison with different parameters**

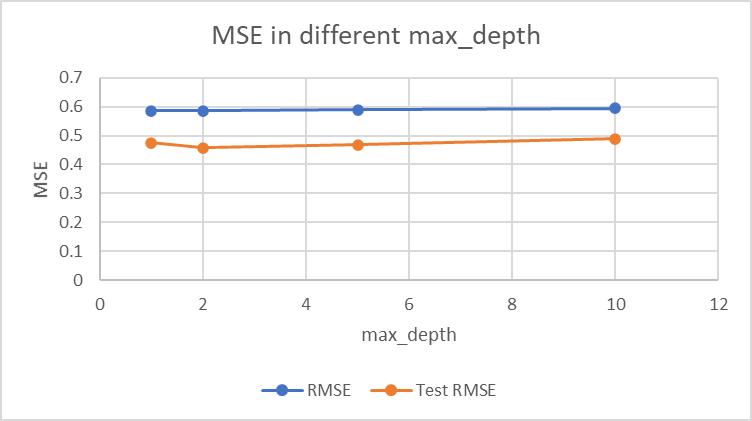
In order to find out how different parameter value affect the model, we need to test the MSE in the testing dataset with different parameter. We decide to test the performance of different learning rate, max depth and also the test size.

First for the learning rate, we find that it does affect the performance of the model. We use the following data for testing and it is very interesting that it gets the best result in learning rate 1. However, since the testing MSE was not affected by the learning rate starting from 0.1 and up, we believe that 0.1 should be the optimal learning rate in our dataset.



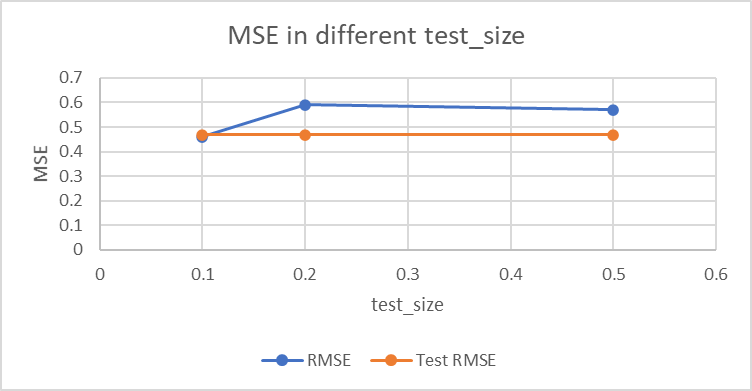
**Fig. 4**

The second parameter we want to compare is the max depth. As shown in figure 5, the max\_depth parameter doesn’t affect the training MSE so much but has a little bit impact in the testing MSE.



**Fig. 5**

The third parameter we want to test is the test size, we believe that different training size and testing size should also affect the outcome. However, after testing different test size in huge variance, the result beat our assumption. As show in figure 6, there remain no change for the testing MSE no matter how the training size change from 90% to 50% and even the training MSE hugely affected.



**Fig. 6**

**Comparison with AdaBoost**

To see how other machine learning model compares with XGBoost, we have examined the AdaBoost, which is a fundamental boosting algorithm. We apply the same dataset with Scikit-Learn library which has the AdaBoost regressor built-in. We adopted the similar cross-validation hyperparameters and scoring parameters, as follows:

cv = RepeatedKFold(n\_splits=10, n\_repeats=3, random\_state=123)

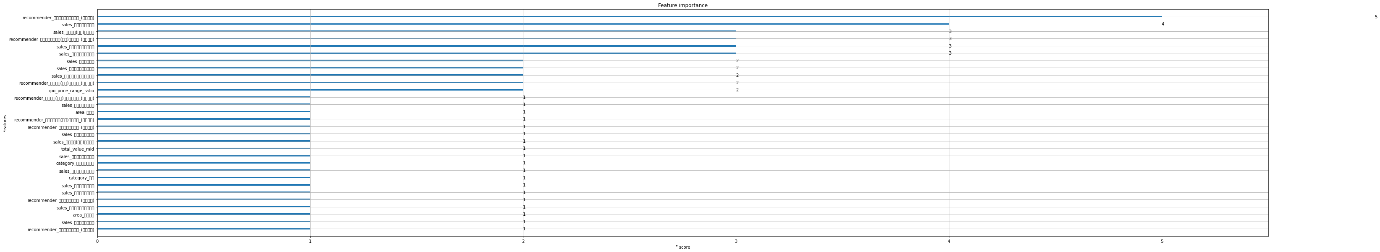
n\_scores = cross\_val\_score(model, X, y, scoring="neg\_root\_mean\_squared\_error", cv=cv, n\_jobs=-1, error\_score="raise")

The rmse obtained from AdaBoost is 0.48368, which is slightly higher than our XGBoost models above. Such difference is small as compared to our XGBoost training can be attributable to the fact that our dataset does not contain large amount of attributes.

Results of the AdaBoost training are saved in the Jupyter Notebook “Ada-learner1.ipynb”.

**Comparison with the Crownpku Repo**

Crownpku Repo contains a sophicated dataset with significantly more number of attributes and are ready for our XGBoost training model to learn. The dataset (with a filename “hk\_ipo\_feature\_engineered”) contains 534 attributes and 278 tuples (as compared with our 9 attributes and 416 tuples). Since the dataset contains a large number of attributes, we have not scrutinize the reasonableness of attributes in detail. Nevertheless, when we apply our XGBoost model and the same set of hyperparameters (in “xgb-learner1.ipynb”), the training suggests a much lower rmse at 0.29024 than our simple dataset. It may suggest that more attributes may help improve the overall accuracy. The following figures shows the important features idenfitied during the training process:



**Fig. 7**

**Future Improvements**

The XGBoost is a powerful learning tool capable of learning small to medium structured/tabular data. However, XGBoost does require considerable efforts in tuning the hyper-parameters as detailed in this report above. Choosing the suitable hyper-parameters may have significant impact to the overall model configuration and hence the final rmse. We suggest that future efforts should be spent on studying the configuration of XGBoost hyper-parameters and the optimisation of the model.

Due to our limitation in obtaining data, which is usually not free in the financial market, we obtained free but limited data from AAStocks.com, as this project is not meant to provide sophisticated results for making a living in investing IPO stocks. It is suggestable for future works to include more data, such as financial figures of the IPO stocks, common financial metrics (e.g. P/E ratio, free cash flow and liquidity ratios), etc.

**Conclusion**

In this project, we successfully implemented today’s most successful machine learning algorithm, the XGBoost model. We have demonstrated it with predicting the first day return of IPO stocks in Hong Kong. An up-to-date dataset containing 416 records and 9 attributes has been obtained from AAStocks.com for our demonstration using XGBoost algorithm. Root-mean-squared-error of our learning model was approximately 0.4694, which suggests that XGBoost is a better prediction model than AdaBoost, which trained the same dataset and obtained an RMSE of 0.4837. We are of the view that if we can successfully identify more relevant data attributes, the difference of performance may be more significant. Our XGBoost model has also trained a large and more sophisticated dataset obtained in the Crownpky Repo, which has 534 attributes and 278 tuples. The RMSE obtained from this dataset was significantly better, at 0.2902.

**Code Resources**

All our source codes, dataset and results are included at [HKBU-DavidLo/COMP7015-AI-Project: Files for the AI Project (github.com)](https://github.com/HKBU-DavidLo/COMP7015-AI-Project).

**References**

1. T. Chen and C Guestrin. XGBoost: A Scalable Tree Boosting System. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. August 2016. Pages 785-794.
2. Luque, C., Quinlana, D., & Isasi, P. (2012). Predicting IPO underpricing with genetic algorithms. International Journal of Artificial Intelligence, 8(S12), 133-146.
3. Quintana, D., Sacz, Y., & Isasi, P. (2017). Random forest prediction of IPO underpricing. Applied Sciences, 6(7).
4. B. Baba, G. Sevil. (2020). Predicting IPO initial returns using random forest. Borsa Istanbul Review 20-1(2020) 13-23.
5. S. Russell, P. Norvig. (2009). Artificial Intelligence: A Modern Approach. Prentice Hall Press. 3rd Edition.
6. Documentations of XGBoost (xgboost.readthedocs.io/
7. [www.datacamp.com/community/tutorials/xgboost-in-pyth](http://www.datacamp.com/community/tutorials/xgboost-in-pyth)on
8. [www.geeksforgeeks.org/xgboost-for-regression](http://www.geeksforgeeks.org/xgboost-for-regression)
9. https://github.com/crownpku/hk\_ipo\_prediction