**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 are 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 other than 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. This study is motivated to apply XGBoost to learn first day IPO return in the Hong Kong Stock Exchange (SEHK).

**Input and Output**

A dataset of IPO stocks over the last three years with selected a number of select attributes is obtained from AAStocks.com, which is a leading Hong Kong financial market real-time data provider. Table 1 below provides the attributes which we selected from AAStocks.com and employed for our XGBoost model.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Ticker | Date of listing | | Market capitalisation | | Range of offer price | | Final offer price | | Over subscription ratio | | Allotment odds | | Return on first day |

Table 1

**Brief introduction of XGBoost**

Boosting is the most widely used ensemble learning methods, where multiple weak learners are trained to solve a learning problem and configured to improve the combined prediction results. Weak learners individually have low variance. They have fast learning but due to the low model complexity, bias associated with them is high too. [Generally, the weak learners are short decision trees 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 regularising overfitting:

obj(θ) = L(θ) + Ω(θ)

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

XGBoost is a tree ensemble model that consists of a set of classification and regression trees (CART) where a CART has a score on each leaf which is the prediction score (some refer it as the similarity score). The prediction scores of each individual tree are summed up to get the final score and the prediction model is in the form:

Where *K* is the number of CARTs, *f* is a function in the set of all CARTs .

In training the trees, the key is to define an objective function and optimise it. XGBoost regularises the structure of the trees and configures the prediction scores of the CARTs. An additive strategy is employed, learning a tree one at a time and adding a new tree next. Therefore, the prediction model at step t is shown in the formula:

For mean squared error to be considered for optimising the objective function, the objective function should be:

where c is a constant.

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

**Dataset and Features**

The target investable universe is Hong Kong stock market and our objective is to examine 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 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 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**

*Library 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), [which is our objective]

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} |

[We also tried to adjust [some of the parameters above] to evaluate the impact to convergence. The results are:]

**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.

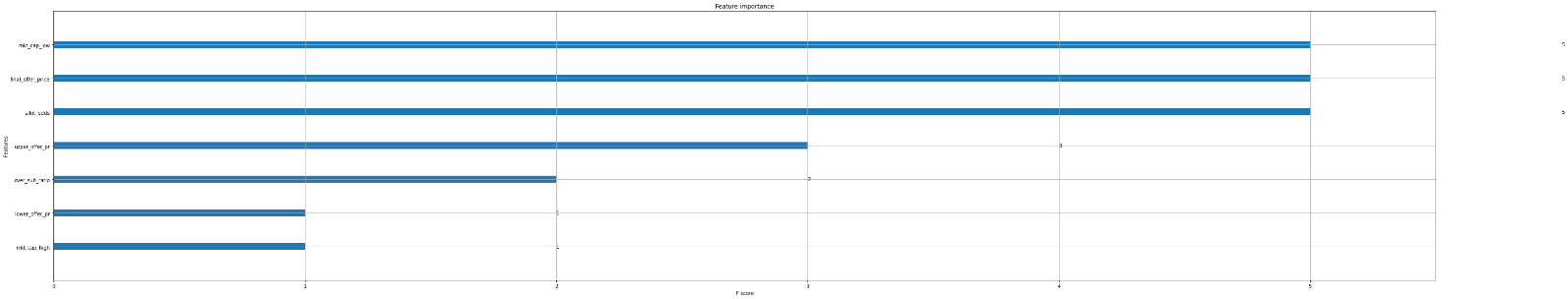
It is a general practice to do a k-fold cross validation to improve the model, where all the entries in the original dataset are used for both training as well as validation, and all entries are used just once. K-fold cross validation is supported by Python’s xgboost using a cv() method. To do this for 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 in our Jupyter Notebook is shown below:



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.

**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**

**Code Resources**

**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