Firms Industry Sector Classification from Quarterly Financial Fundamentals

Bingnan Huo

Data Science Initiative, Brown University

GitHub Project Link[8]

1. Introduction

1.1 Background, Dataset, and Model Formulation

This data science project explores a unique approach to understanding industry sectors through financial data. Using the Compustat® Financial Fundamentals Quarterly dataset[1], which covers a wide range of financial information for companies in the world, I hope to classify industry sectors based on trends in firms' earnings. Previous studies have often focused on using sector data to predict future earnings, but this project aims to show the connection from using a reversed approach: if we can accurately classify industry sectors by looking at their financial data, then we can also infer a company's profitability and earnings potential simply by knowing its industry sector.

Given the nature of the quarterly fundamentals data, our dataset follows a grouped and time-series structure – each firm contains multiple rows, representing their fundamentals for each quarter from 2000Q1 to 2023Q3. Moreover, given my goal for sector classification, this is a multi-class classification problem. I will use earnings fundamentals, with feature engineering, to classify firms into 1 of 11 sectors. The Global Industry Classification Standard (GICS®)[2] was used as the standard for top level sector classification.

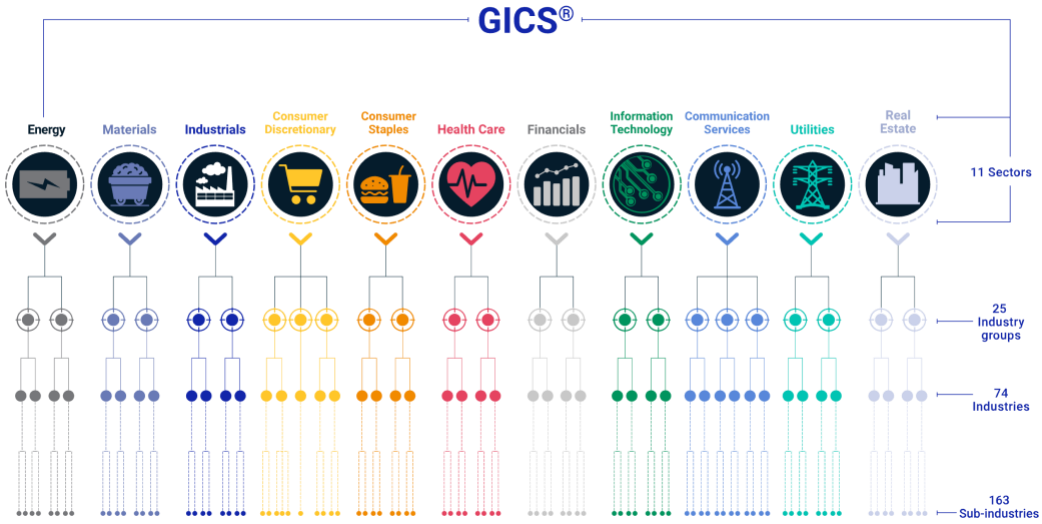

Fig 1. The top levels of sector classification are used as the response variable.

1.2 Data Cleaning and Feature Engineering

My dataset was acquired through UPenn Wharton Research Data Services[1], but in two datasets: North America region and Global region. I first prepared the dataset by aligning column names, removing unused columns, removing firms with completely missing earnings data, limiting the year past 2000, and finally combined the two datasets into one. The resulting dataset contains more than 2.7 Million rows and 20 columns, with 58930 unique companies across 11 sectors.

These features include time (quarter), currency, revenues, costs of goods sold, and various variables such as earnings, cashflow, assets, debts, etc. Next, I conducted feature engineering and created 15 financial ratios from combination of existing variables, guided by domain knowledge. These ratios, including gross profit margin, asset to liability ratio, etc., were cap at 2 to prevent ratios that are unrealistically high or low, due to dividing by a small number.

Furthermore, I hand-picked 13 of my variables (including ratios) to create 4 lagged- and 8 sliding-window mean and standard deviation variables, each tracking trends of the picked variables of past 4 or 8 quarters. After completing feature engineering, the dataset now contains 139 columns. However, creating lagged and sliding-window variables also resulted in many missing values.

Full descriptions for variables and feature engineering can be found in the data folder of the project.

2. Exploratory Data Analysis

We now being EDA to have a first look at the response variable. From Fig 2, notice that we have serious class imbalances, with the most populous class Industrials having around 18% and the minority class Real Estate having only around 2%. In an even distribution, each class would represent roughly 9% of the data. Thus, we have to be aware of this in the future when we fit models to data, especially when splitting the data, to avoid too few observations of the minority class in train, validation, or test set.

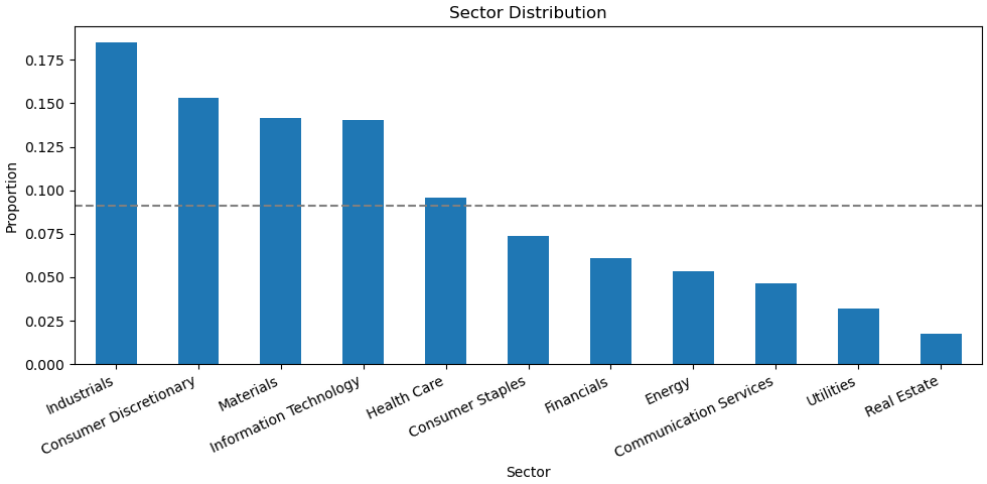

Fig 2. The sector distribution shows serious class imbalances in the data, dashed line represents even distribution.

Next, we will sneak peek at our explanatory variables to understand relationships between our feature set and the sectors. As I learned from domain knowledge[3], profitability is often the most characteristic aspect of an industry sector. Fig 3 shows the Operating Profit Margins across sectors over the years. It's interesting to see that, even if some sectors are similar average profit margins, their trends can be different. For example, the top two lines are Financials and Utilities sectors, and we see Utilities (light blue line) shows margins with more seasonality each year and see margins dropped significantly in the most recent quarters (2023Q2, Q3), potentially due to inflation and interest rates. Fig 4 shows the same information, but is easier to see individual sectors' margin trends and variablities.

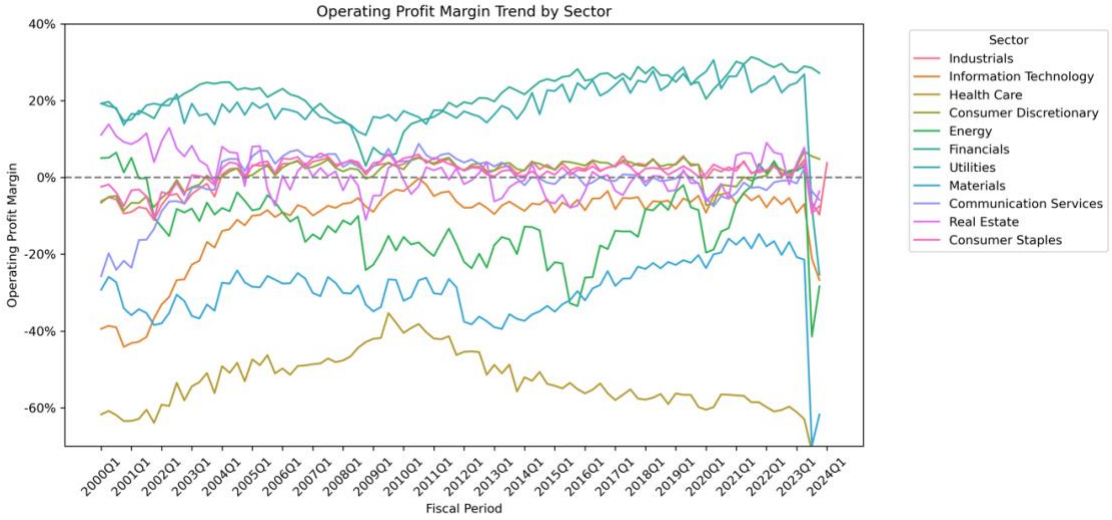

Fig 3. Operating profit margins by sector, sectors demonstrate different trends over past 20 years.

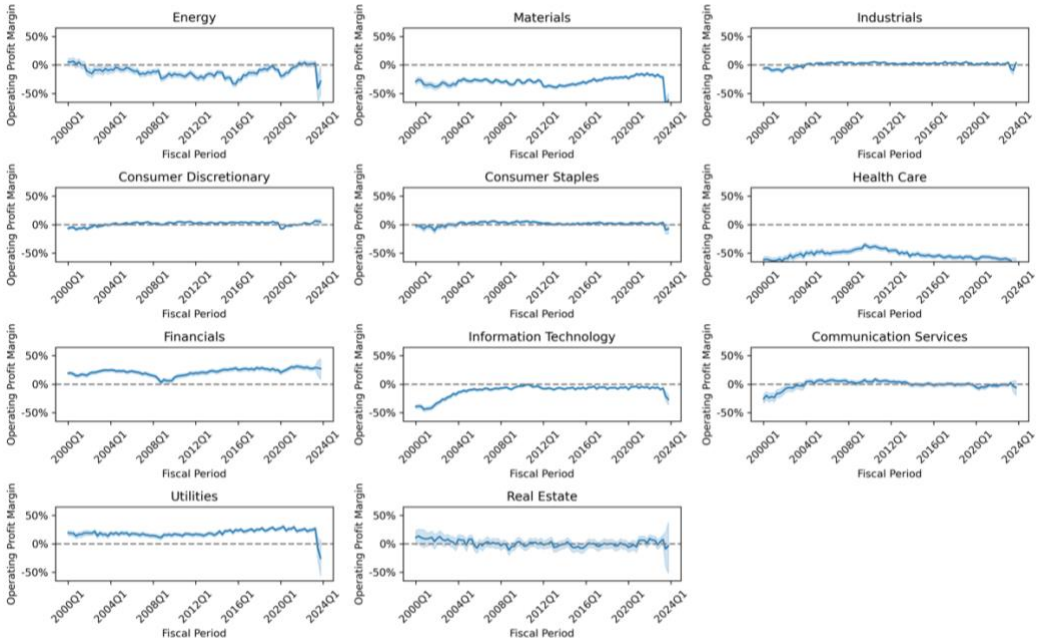

Fig 4. Operating profit margins by sector, sectors show different variabilities as well.

3. Methodology

Given our problem being multi-class time-series classification, with a lot of missing values, preliminary research[4] shows that in fact standard machine learning algorithms, including the models we learned at class this semester, are not well suitable for the task.

The primary reason is that the independence assumption of the data is being violated, since the time and group structure certainly implies dependence between rows of data. Moreover, these algorithms cannot best make use of the sequential nature of our data; for example, rows for a firm across quarters, should be considered as one firm for classification. However, algorithms such as XGBoost, Random Forest, Support Vector Machine, and Logistic Regression would treat each row as a single firm (while assuming independence between firms) for classification.

What makes this problem even more challenging, is the fact that a lot of missing values are present in our dataset, due to data reporting, collections, and the lagged and sliding-window variables I have created. These algorithms, except XGBoost, cannot fit to data with missing values. Lastly, while the huge dataset size represents opportunities and information value, it also creates problems for model fitting.

In this project, I used XGBoost 2.0.2[5] and Scikit-learn 1.3.0[6].

3.1 Splitting

3.1.1 Train-Test Split

With all these challenges in mind, I will proceed with splitting. First, I use *StratifiedGroupKFold* to split the data to obtain an 80-20 split between train (& val) and test groups, while maintain group structure (firm) and making sure that the distribution in response is roughly the same as the down-sampled set (even) by stratifying.

3.1.2 Down-sampling

Noting the class imbalances in our dataset, I wrote a custom function to identify the minority class and down-sample all other classes to the same size as the minority class. The group structure is also considered when sampling. I performed down-sampling only on the train-val set, so test set response distributions would remain similar. The down-sampled size is roughly 20% of the train, with the minority kept at the same size, but larger classes reduced significantly. In Fig 5, we see that our dataset is now balanced.

Down-sampling was also necessary for training time purposes, even after down-sampling to 1%, our train set still has 20,000 plus rows. Hyperparameter tuning via search and CV will take days if we use the full dataset.

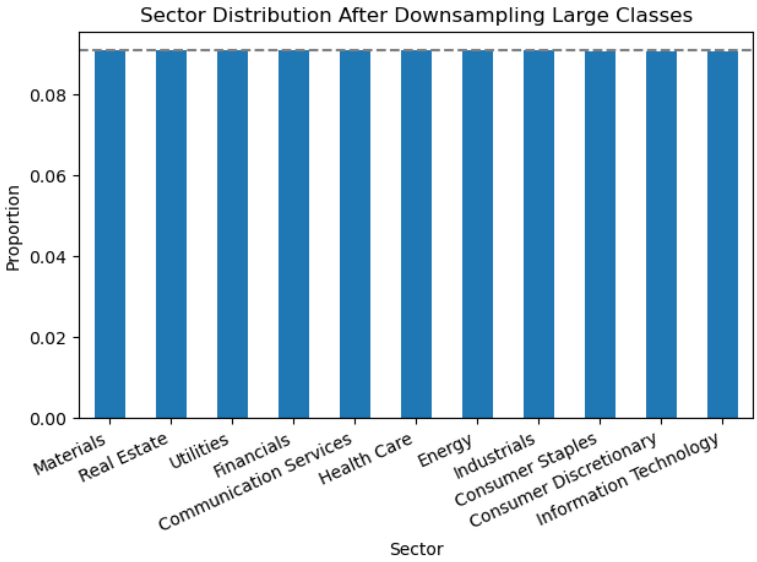

Fig 5. Down-sampling reduces large classes to have similar size to the previous minority class

3.2 Preprocessing

3.2.1 Categorical Variable Encoding and Feature Scaling

In preprocessing, the first task is to encode categorical variables with one-hot encoding and scale all numerical variables (including original and engineered features) with standard scaler. This will best prepare our dataset for numerical stability.

However, to deal with missing values, I opt for imputing them with 0 after feature scaling, which is equivalent to imputing by column mean. I don't want to remove columns with many missing values directly, since this will eliminate most of sliding-window variables I created. Yet, for XGBoost, where I used dataset containing missing values since it can handle missing values well.

3.2.2 Dimensionality Reduction

To alleviate the impact of imputing on our dataset, I applied Principal Component Analysis to re-represent my data, to reduce the number of columns while retaining 99% of variance. This will also lead to faster training time.

Imputing also made applying PCA possible, since it does not accept missing values. Training XGBoost models did not involve PCA, since the model could deal with missing values.

3.3 Evaluation Metric

As discussed earlier, a firm will have multiple rows of data and multiple predictions. To address this problem, I wrote a voting function that aggregate the results across rows. In Table 1, all the rows shown represent a single firm's record over the years. Note that classifications here are generated by one model, we are not voting across models.

Company	Date	Actual Labels	Predicted Labels	Voted Label
Firm A	2010Q1	Sector 1	Sector 2	**Sector 1**
Firm A	2010Q2	Sector 1	Sector 2	
Firm A	2010Q3	Sector 1	**Sector 1**	
Firm A	2010Q4	Sector 1	**Sector 1**	
Firm A	2011Q1	Sector 1	**Sector 1**	
…		…	…	…

Table 1. Illustration of the voting function.

Considering class imbalances, I decided to use the weighted f1 score. Thus, accuracy and f1 scores were also calculated on the voted predictions.

3.4 Hyperparameter Tuning and Cross Validation

For hyperparameter tuning, I opt to use *BayesSearchCV*, which was offered by the scikit-optimize package[7]. It effectively employs Bayesian optimization to selects parameters that are most likely to improve model performance, in very few iterations. If using *GridSearchCV*, hundreds of iterations are required, whereas *BayesSearchCV* only took 25.

Ideally, I would also have the whole search process loop over several different random states. However, due to training time and resources, I could only run the process once.

Model	XGBoost	Random Forest	Support Vector Machine	Logistic Regression
Parameters Searched	Learning_rate (0.1-0.5), n_estimators (200-300), max_depth (7-12), colsample_bytree (0.3-1), Subsample (0.3-1), Alpha (0.1-10), lambda (10,100)	n_estimators (100-300), max_features (sqrt, log2), max_depth (3-35), min_samples_split (3-50)	kernel (linear, rbf, sigmoid), C (1e-3 – 1e3), gamma (1e-4 – 1e3)	penalty=elasticnet C (1e-4 – 1e4), l1_ratio (0-1), max_iter (5000-10000)
Sample Size	5% (137K+)	4% (110K+)	0.3% (8K+)	1% (27K+)
PCA (Variance Explained)	100% (No PCA)	99%	95%	95%
Optimal Parameters	learning_rate=0.1 n_estimators=300 max_depth=12 colsample_bytree=1 subsample=0.3 alpha=0.5, lambda=100	n_estimators=300 max_features=sqrt max_depth=31 min_samples_split=2	kernel=rbf C=3 gamma=0.0039	C=0.0145 l1_ratio=0 max_iter=10000
F1 score	44.5%	33.8%	22.2%	21.1%

Table 2. Models and hyperparameters searched, optimal parameters and classification f1 score also shown.

4. Results

4.1 Model Performances

XGBoost was the best model, with a weighted f1 score of 0.445 after classification voting. Moreover, across different models, most showed that voting improves classification score.

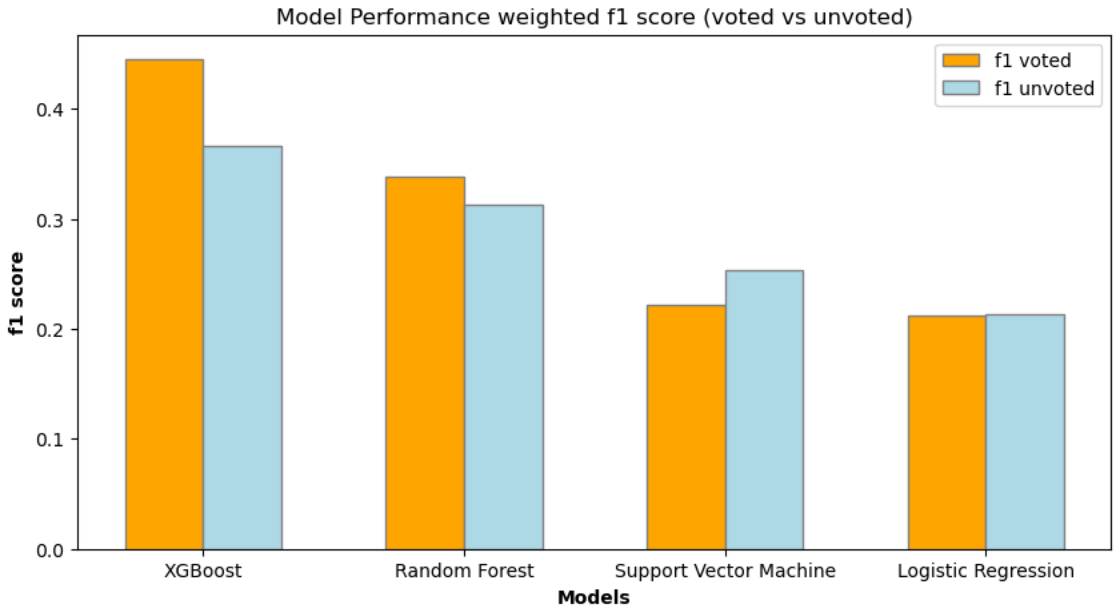

Fig 6. Performance between models tried, showing comparison between voted and unvoted predictions.

4.2 Best Model

With XGBoost identified as the best model and acquired hyperparameters, I then refit the model with the full dataset. I also used early stopping (150 rounds) by providing the model with a validation set. Note that all numbers are voted and weighted results.

Accuracy	Precision	Recall	F1 Score
47.5%	50.4%	47.5%	46.6%

Table 3. XGBoost performance metrics.

Looking at the confusion matrix, we can also see that the model is not doing bad! Given there are 11 total classes. Do note that the model tends to classify firms into sector 2 (which is industries). This might be due to class imbalance (as the final model was trained on full dataset) or inherently similar trends between sectors. Overall, we see diagonal squares having deeper colors, showing successful classifications.

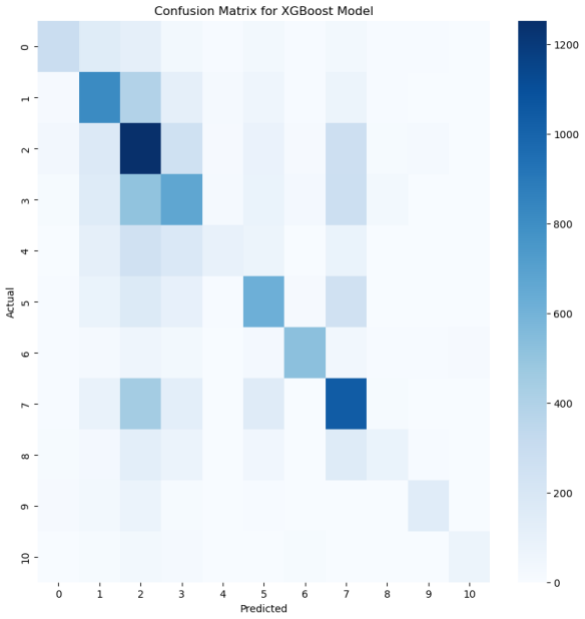

Fig 7. Confusion matrix for final XGBoost model.

4.3 Feature Importance

Looking at XGBoost model global feature importance, I think total gain is the most useful. We see that the model successfully used engineered features such as gross profit margin, operating profit margin, and total turnover rate (*gpm, opm, tat*). Moreover, sliding window variables that track turnover mean and std across 8 quarters also showed up, meaning "trends" discussed earlier are also important.

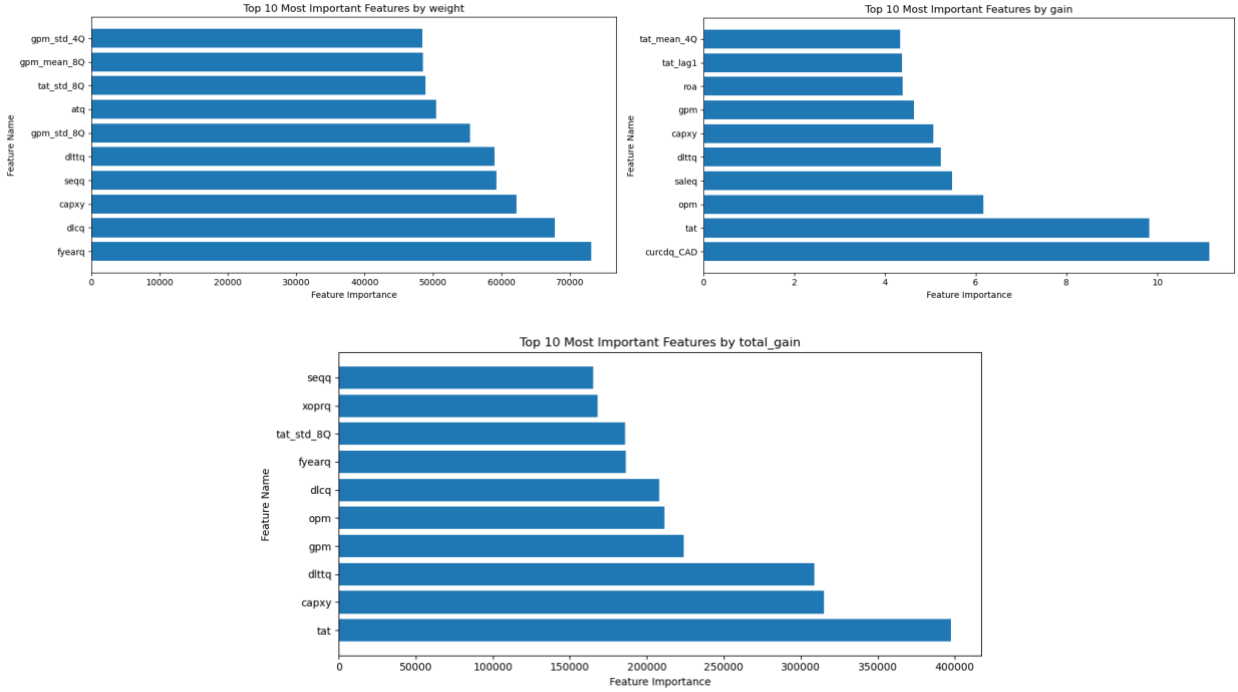

Fig 8,9,10. XGBoost global feature importance measured by weight, gain, and total gain.

5. Outlook

To conclude my project, I believe my best model achieved moderate performance with 45% f1 score (weighted and voted). If I had more time and computation resources available, I would train and evaluate models with the full dataset across several random states to better understand model performance stability.

As I worked through the project, I learned that, in fact, the most suitable algorithms would be Neural Networks, such as Recurrent Neural Network or Convolutional Neural Network. Thus, in the future, I would love to try this modeling problem again with Deep Learning and Neural Networks, so the models can be effective in using the sequential nature of our data.

6. References

[1] Standard & Poor's (n.d.). *Compustat daily updates – fundamentals quarterly* [All companies listed: Income statement items - January 2020 to October 2023]. [Data set] CapitalIQ. Retrieved October 15, 2023, from Wharton Research Data Services. https://wrds-web.wharton.upenn.edu/

[2] MSCI Inc. (n.d.). *The Global Industry Classification Standard (GICS®)*. GICS® - The Global Industry Classification Standard - MSCI. Retrieved October 15, 2023, from https://www.msci.com/our-solutions/indexes/gics.

[3] "Profit Margin by Industry." FullRatio. Retrieved October 15, 2023, from https://fullratio.com/profit-margin-by-industry.

[4] Enes. (2022, April 12). How to use time-series observations on multi-class classification problem. Stack Exchange. https://stats.stackexchange.com/questions/571307/how-to-use-time-series-observations-on-multi-class-classification-problem

[5] Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (pp. 785–794). ACM.

[6] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Dubourg, V. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12, 2825-2830.

[7] Head, T., Kumar, M., Nahrstaedt, H., Louppe, G., & Shcherbatyi, I. (2021). scikit-optimize/scikit-optimize: v0.9.0 (Version v0.9.0). Zenodo. DOI: 10.5281/zenodo.5565057

[8] https://github.com/BingnanHuo/DATA1030_Project