DATA 1030 Midterm Project Report

Bangxi Xiao

Data Science Initiative, Brown University

Code base: *https://github.com/rchopinw/DATA1030_MIDTERM_PROJECT*

1. Introduction

1.1. Motivation

The peer-to-peer lending has grown rapidly in recent years and provides loans to serve many purposes, such as consumer credit, small business and student education. The motivation behind our project is to help lenders make better lending decisions to maximize returns while minimize risks. This project provides critical application for investors to predicting future loan status using the Kaggle dataset.

1.2. Data Overview

The dataset is obtained from Kaggle website: https://www.kaggle.com/mishra5001/credit-card, which has a size of over 100 features and 300,000 samples. According to the column "TARGET", the data could be split into two parts – the defaulters and the non-defaulters, where defaulters are identified as people who are rejected by the loan company while the non-defaulters are accepted by the companies. The features include personal information such as number of children, credit amount and so on. Also, some third-party credit rates are involved.

1.3. Goal

The first step of the project is to successfully process the missing values – some of the features suffered from large proportion of missing values and we have to deal with them. Next, we are going to resolve the problem of unbalanced data and the curse of dimension problems. The ultimate goal of the project is to successfully identify the defaulters given their information (a binary classification task).

1.4. Related Work

The data was obtained from Kaggle and a few previous works were done by the contributors on Kaggle. Their efforts were mainly concentrated on binary classification problem, exploratory data analysis and data structure analysis. Joe Corliss [1] made prediction on loan charge offs from initial listing data and proposed a logistic regression with SGD parameter training algorithm. Also, he performed some other models such as random forest classifier and k-nearest-neighbor.

2. Exploratory Data Analysis

2.1. Target Variable

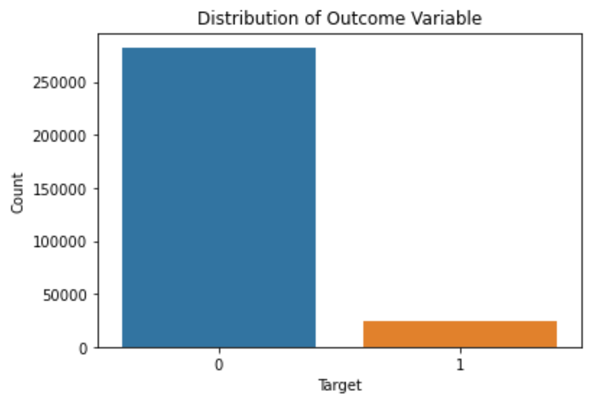

Fig 0: The number of defaulters vs non-defaulters, the samples are highly unbalanced.

The target variable – whether a loan is default or not, as we can see, is highly imbalanced. 0 indicates the non-default status while 1 indicates the default status.

2.2. Selecting from Similar Features

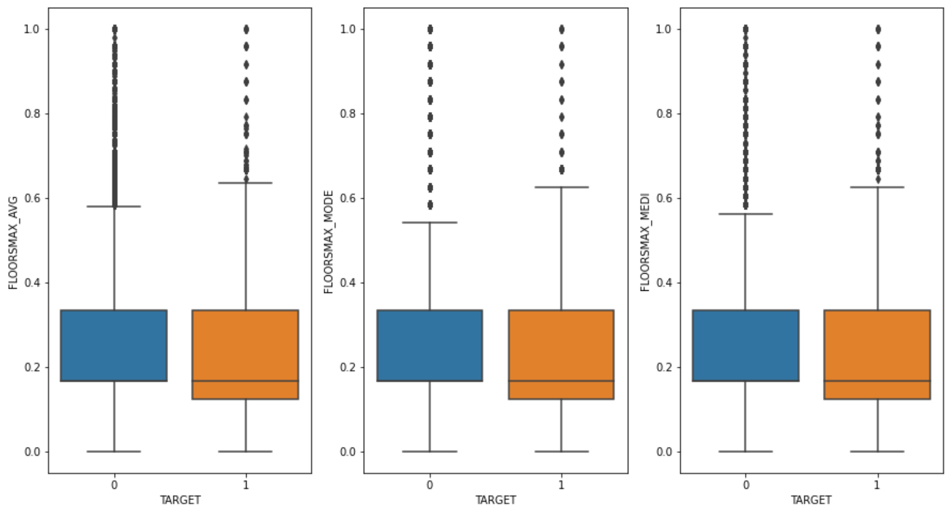

Fig1: Floor-related features refer to the number of floors (normalized) where the client lives has, where AVG indicates the average number of floors, MODE the maximum and MEDI the median. We can see that there are some slight differences between defaulters and non-defaulters: the defaulters have most of the floors lower than the non-defaulters.

2.3. Removal of Non-informative Features

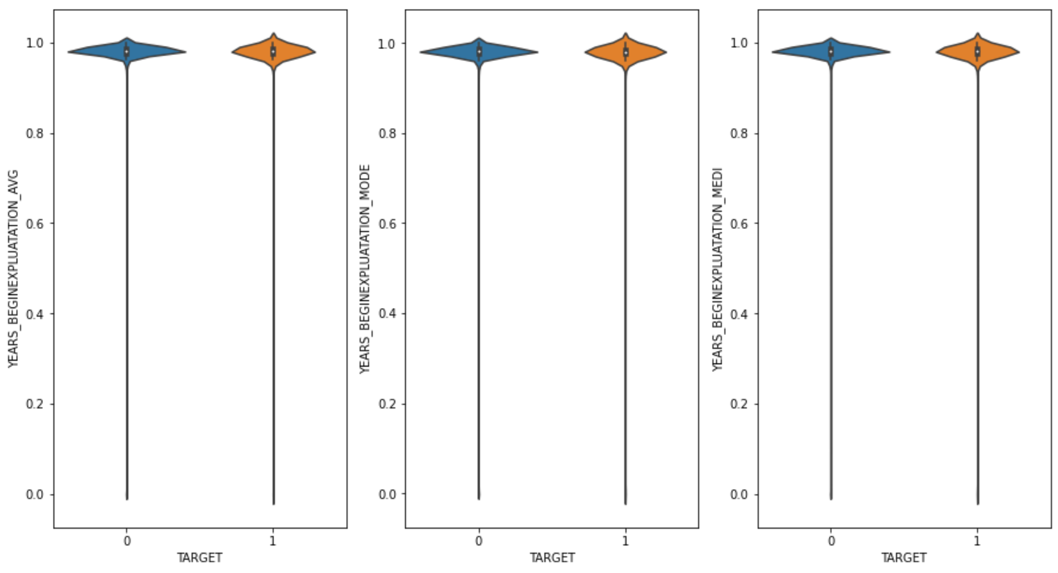

Fig 2: We can hardly tell the difference from the 3 features between defaulters and non-defaulters since their distributions are too close to each other. We might consider delete the 3 features from our model.

2.4. Linear Relation Stud

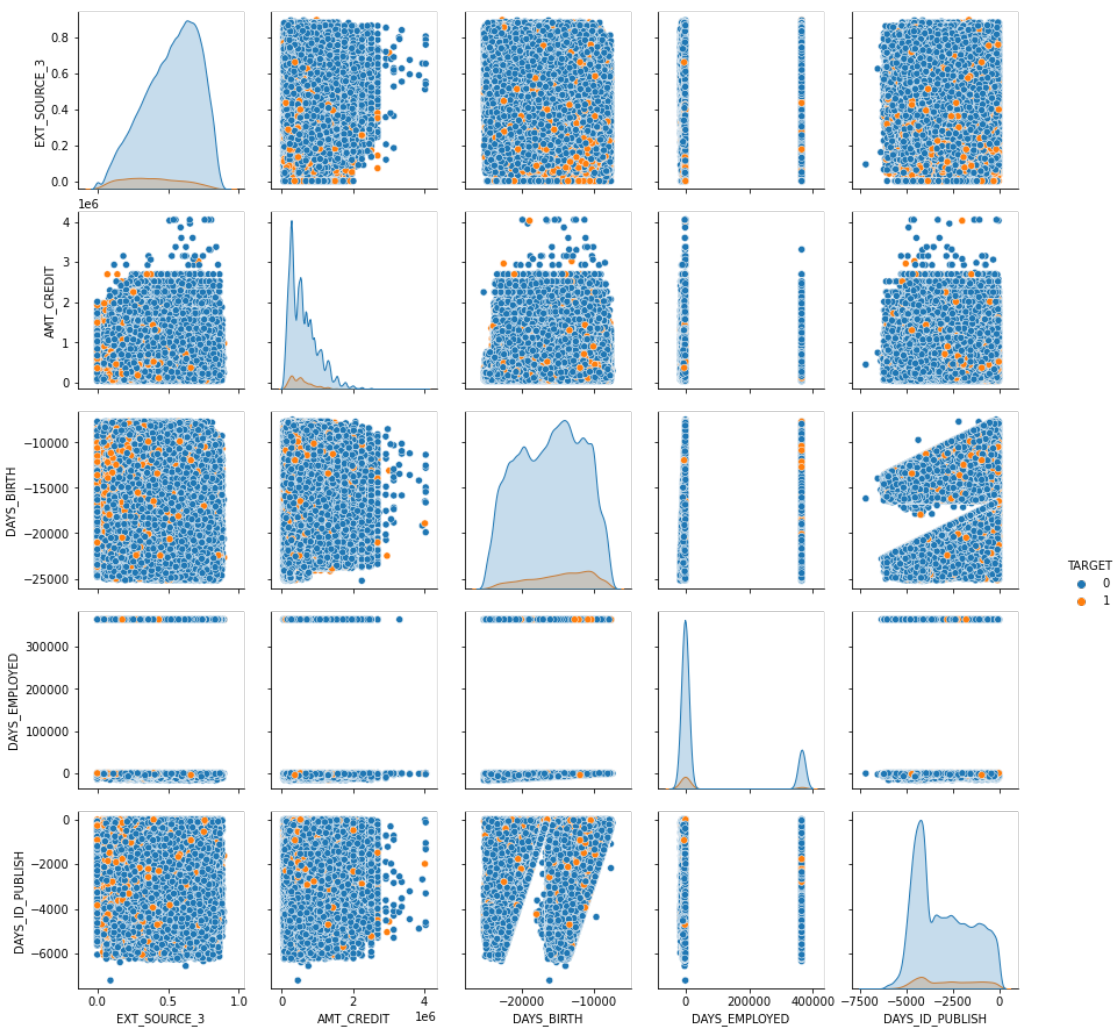

Fig 3: We examined a selection of continuous features by plotting the density curves and scatter plots. To see the discrepancies between defaulters and the non-defaulters.

2.5. Categorical Feature Example

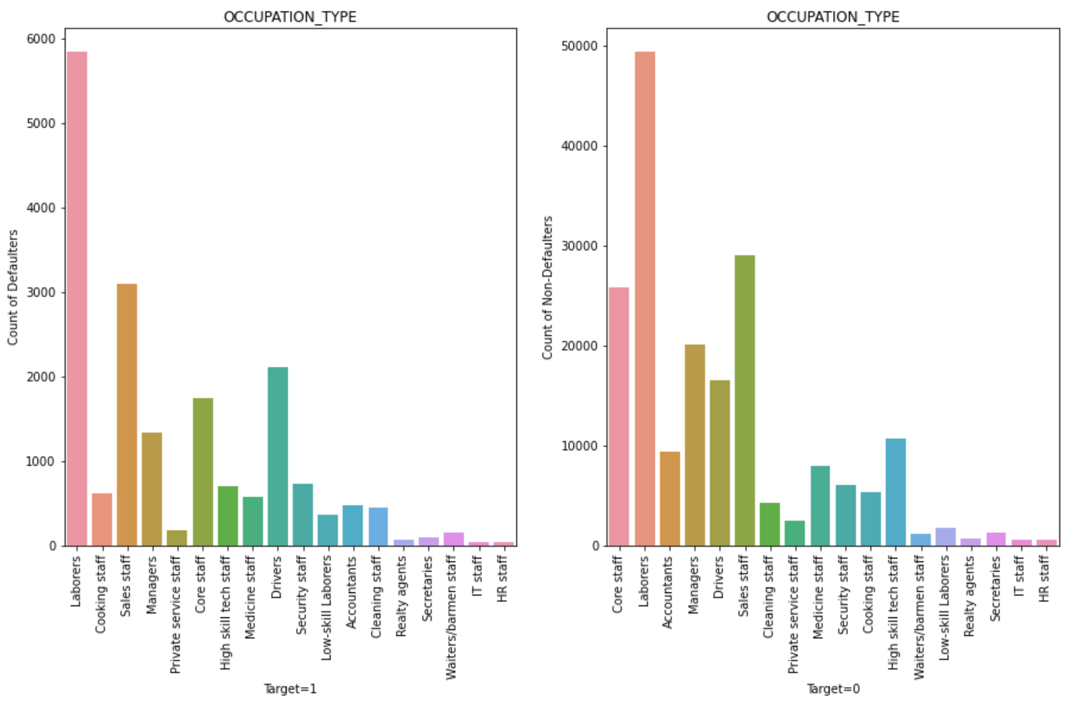

Fig 4: This figure illustrates the customers' occupations. Some discrepancies are observed between the defaulters and the non-defaulters in this plot, for example, the number of managers who claim to default a credit is lower than the non-default ones.

3. Methods

3.1. Data Preprocessing

3.1.1.Handling Missing Value

Since different models have different requirement for the missing values. We adopted two ways to handle the missing values:

We first filtered out the features with missing proportion greater than 50.00%. For features with missing proportion less than 10.00%, we directly ruled out the corresponding rows.

In categorical features, we used “Other” and “Unknown” to separately impute the missing values in “OCCUPATION_TYPE” and “EMERGENCYSTATE_MODE”, before which, we have checked that the data is missing at random.

For the missing values that are identified as continuous, we used iterative methods to do the imputation; for categorical missing values, we created a new category “missing” to indicate that the values are originally missing.

Secondly, for the XGB model, since it is able to deal with the missing values with its unique model structure, we simply imputed the missing values in the continuous features with “NaN” value and categorical features with “missing” value.

3.1.2. Generic Preprocessing Steps

Ordinal features are treated with ordinal encoder; continuous features are treated with standard scaler; categorical features are treated with one-hot encoder. We processed the data in multiple ways and the number of features in each dataset is different:

	Dataset Version	**Categorical Features**	**Continuous Features**	**Ordinal Features**	**Imputation Technique**
Data Unimputed	Ver. 1	42	32	4	N/A
	Ver. 2	20	41	4	N/A
Data Imputed	Ver. 1	40	17	4	Iterative
	Ver. 2	21	23	2	Constant

Table 1: Data Preprocessing Results

3.1.3. Splitting Strategy

A stratified splitting should be performed. The dataset is I.I.D. since each row represents each client and the clients are different from each other. Also, it does not contain any grouping structure or time-series pattern. Under each random state, the data is split into three parts – training, validation and testing, with proportion of each 0.8, 0.1 and 0.1.

3.1.4.Data Balancing

The major approach is to adjust the class weights; by increasing the weight of the minor class samples and decreasing the weight of the major class samples, the loss-based machine learning models (logistic regression, tree models and boosting models) are able to reconstruct the loss function into a weighted one. As a result, the loss in minorities is less tolerable comparing to the majorities. It is more stable comparing to SMOTE.

3.2. Machine Learning Pipeline

In this section, we will reveal more detail regarding the model frameworks. All the models in our report used the same training pipeline below:

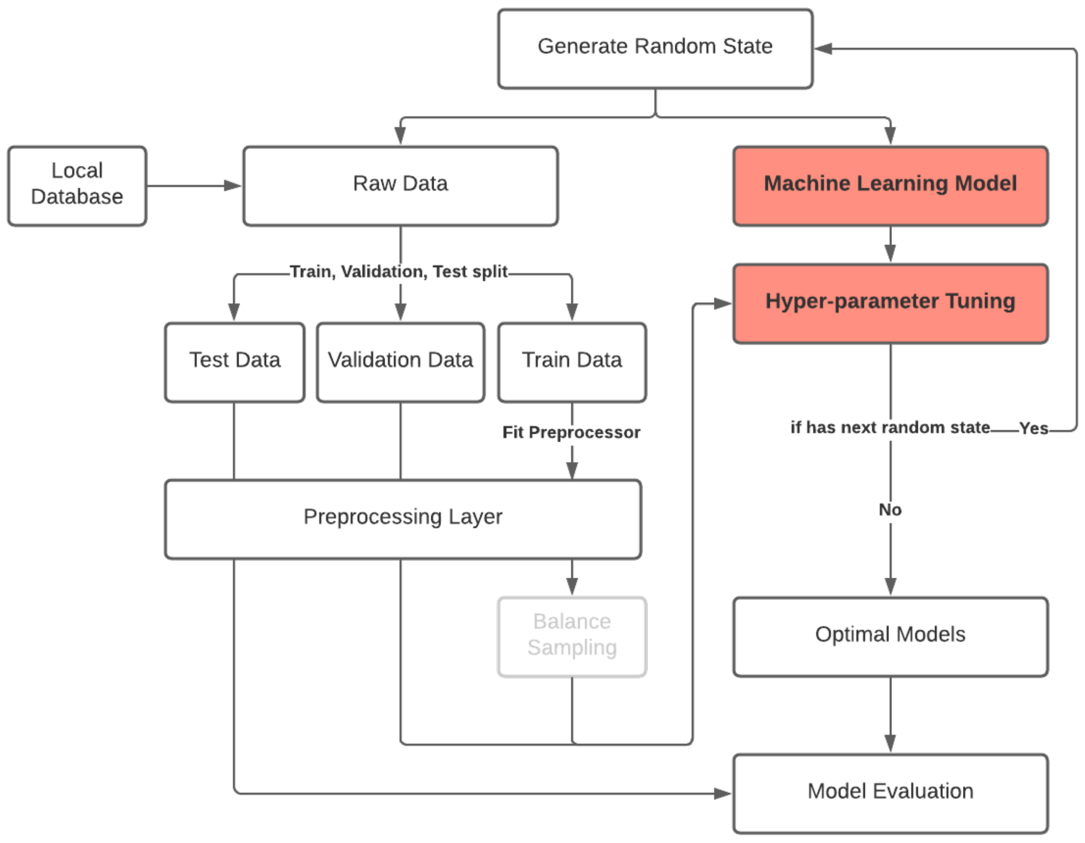

Figure 5: Machine Learning Model Training Pipeline

The above workflow is the complete pipeline of training a machine learning model. As we can see, we set multiple random states (number of 5 in our experiments) for the whole training environment – to make sure that no matter how we split and train the model, the randomness part won't affect the performance of the optimal model; we used this technique to rule out the uncertainties within. Specifically, a fixed set of random states is generated before the start of the training process; then, under each random state, a full loop of training, validation, tuning, selection and evaluation were performed. The output of the above framework includes the optimal hyper-parameter settings and optimal model's test score. We don't expect to see too much variation in different random states.

3.3. Baseline Model

The baseline model, according to the label distribution in our dataset, is predicting all the testing labels as the major class – the non-defaulters. Thus, our baseline model has an overall accuracy of 0.91.

3.4. Logistic Regression

A table of detailed model settings are listed below:

Logistic Regression Model

Class-weighted training	**Dimension reduction**	**Balance sampling**	**Hyper-parameter tuning**	**(train, validation, test)**	**Learning rate**	**optimizer**
Neg/Pos	Truncated-SVD	N/A	C	(0.8, 0.1, 0.1)	0.001	SGD

Table 2: Logistic Regression Model Training Facts

The hyper-parameter "C", which controls the penalty of the logistic loss function, is set ranging from 0.01 to 100, a length of 10.

We used metric "roc_auc" to select the optimal hyper-parameters and evaluate the model performance, which focuses on the performance on both positive and negative classes. Also, other metrics such as accuracy, recall, precision and f1-score are also calculated.

3.5. Random Forest Model

We trained the random forest model using the following parameter settings:

Random Forest Model				
Balancing sampling	**Hyper-parameter tuning**	**(train, validation, test)**	**Class-weighted training**	**Dimension reduction**
N/A	n_estimators max_depth	(0.8, 0.1, 0.1)	Neg/Pos	N/A

Table 3: Random Forest Model Training Facts

The number of estimators is within the following range:

[10, 50, 100, 200, 300, 500]

Similarly, the maximum depth of the tree has the following choices:

[2, 4, 8, 16]

A thorough grid search with metric "roc_auc" will be performed to obtain the optimal parameter combination. We will also calculate other major metrics mentioned before.

3.6. Hist Gradient Boosting Model

The training settings are listed below:

Hist Gradient Boosting Model

Balance sampling	**Class-weighted training**	**Dimension reduction**	**(train, validation, test)**	**Hyper-parameter tuning**	**Learning rate**	**Loss**
SMOTE	N/A	N/A	(0.8, 0.1, 0.1)	max_depth, max_leaf_nodes	0.1	auto

Table 4: Hist Gradient Boosting Model Training Facts

The maximum depth has the following values:

[2, 4, 8, 16, 32]

The maximum number of leaf nodes has:

[4, 8, 16, 32, 64, 128]

Likewise, "roc_auc" is the core metric of the hist gradient boosting model. Other major metrics will also be included.

3.7. Extreme Gradient Boosting Model

The XGB model has the following settings:

Extreme Gradient Boosting Model						
Balance sampling	**Class-weighted training**	**Dimension reduction**	**(train, validation, test)**	**Hyper-parameter tuning**	**Learning rate**	**Tree method**
N/A	Neg/Pos	N/A	(0.8, 0.1, 0.1)	n_estimators, max_depth	0.1	gpu_hist

Table 5: XGB Model Training Facts

The number of estimators is selected from below:

[160, 320, 500, 800, 1600, 2000, 3200]

The maximum depth:

[2, 4, 8, 16]

During the grid search, the "roc_auc" score is used to select the optimal hyper-parameters. Other scores will also be evaluated on the testing set.

4. Results

In this part, we examined the resulting models' overall performance with multiple metrics and evaluated the robustness and stableness of the models.

4.1. Model Performance

Model/Metrics	**Data Source**	**Accuracy**	**Recall**	**Precision**	**F1-Score**
Baseline *(reference)*	N/A	0.9190 (± 0)	0 (± 0)	N/A (± 0)	N/A (± 0)
Logistic Regression	DI-1	0.6013 (± 0.0018)	0.6127 (± 0.0078)	0.1188 (± 0.0011)	0.1990 (± 0.0019)
	DI-2	0.5796 (± 0.0045)	0.6020 (± 0.0056)	0.1115 (± 0010)	0.1881 (± 0.0015)
Random Forest	DI-1	0.6960 (± 0.0013)	0.6409 (± 0.0067)	0.1586 (± 0.0015)	0.2542 (± 0.0025)
	DI-2	0.6902 (± 0.0031)	0.6453 (± 0.0097)	0.1566 (± 0.0021)	0.2520 (± 0.0034)
Hist Gradient Boosting	DI-1	0.9187 (± 0.0003)	0.0252 (± 0.0025)	0.4495 (± 0.0327)	0.0476 (± 0.0046)
	DI-2	0.9184 (± 0.0001)	0.0284 (± 0.0021)	0.4340 (± 0.0131)	0.0534 (± 0.0039)
Extreme Gradient Boosting (XGB)	DI-1	0.7045 (± 0.0018)	0.6707 (± 0.0097)	0.1678 (± 0.0017)	0.2685 (± 0.0029)
	DI-2	0.6998 (± 0.0025)	0.6733 (± 0.0037)	0.1660 (± 0.0007)	0.2663 (± 0.0008)
	DU-1	**0.7060 (± 0.0031)***	**0.6805 (± 0.0063)***	**0.1700 (± 0.0019)***	**0.2721 (± 0.0027)***
	DU-2	0.7046 (± 0.0028)	0.6788 (± 0.0073)	0.1690 (± 0.0017)	0.2707 (± 0.0026)

**The standard deviation is denoted as (±) below the score.*

Table 6: Model Performance Evaluation and Comparison

By comparison, the optimal model we developed here achieved an overall accuracy of 0.7060, approximately 0.2 below the baseline model's accuracy. However, the recall improves for about 0.68, which suggests that the ability of identifying the defaulters is largely increased.

4.2. Feature Importance and Interpretation

In this part, we specifically examined the performance and feature importance yielded from the optimal XGB model trained with dataset "DU-1". First, we calculated the top feature importance with metrics such as weight, gain, cover, total gain and total cover, which are considered as "global importance". Then, we further calculated the local feature importance with metric SHAP value. Visualizations and feature importance ranking will be revealed in this part.

Under the weight metric, the top 20 features are given below:

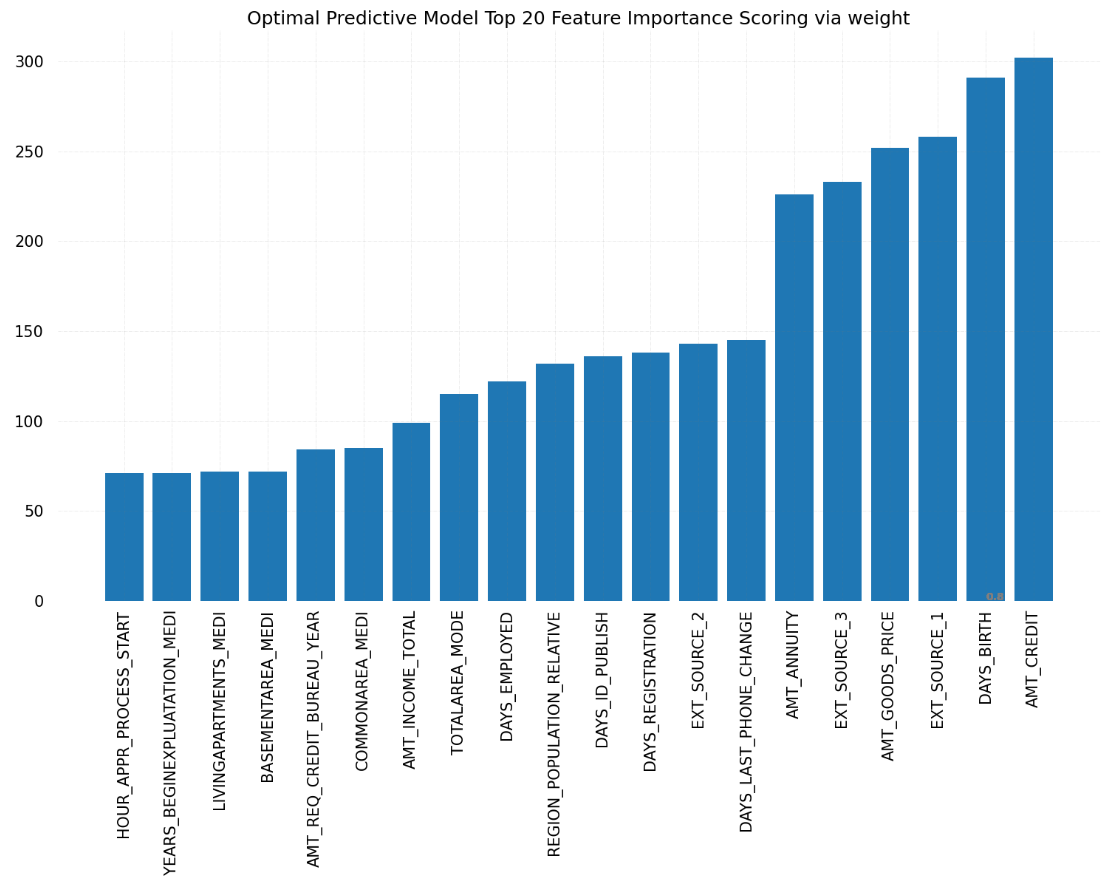

Figure 6: Feature Importance of Optimal XGB Model by Weight

Also, under the metric “gain” and “total gain”, the feature importance scores are visualized as below:

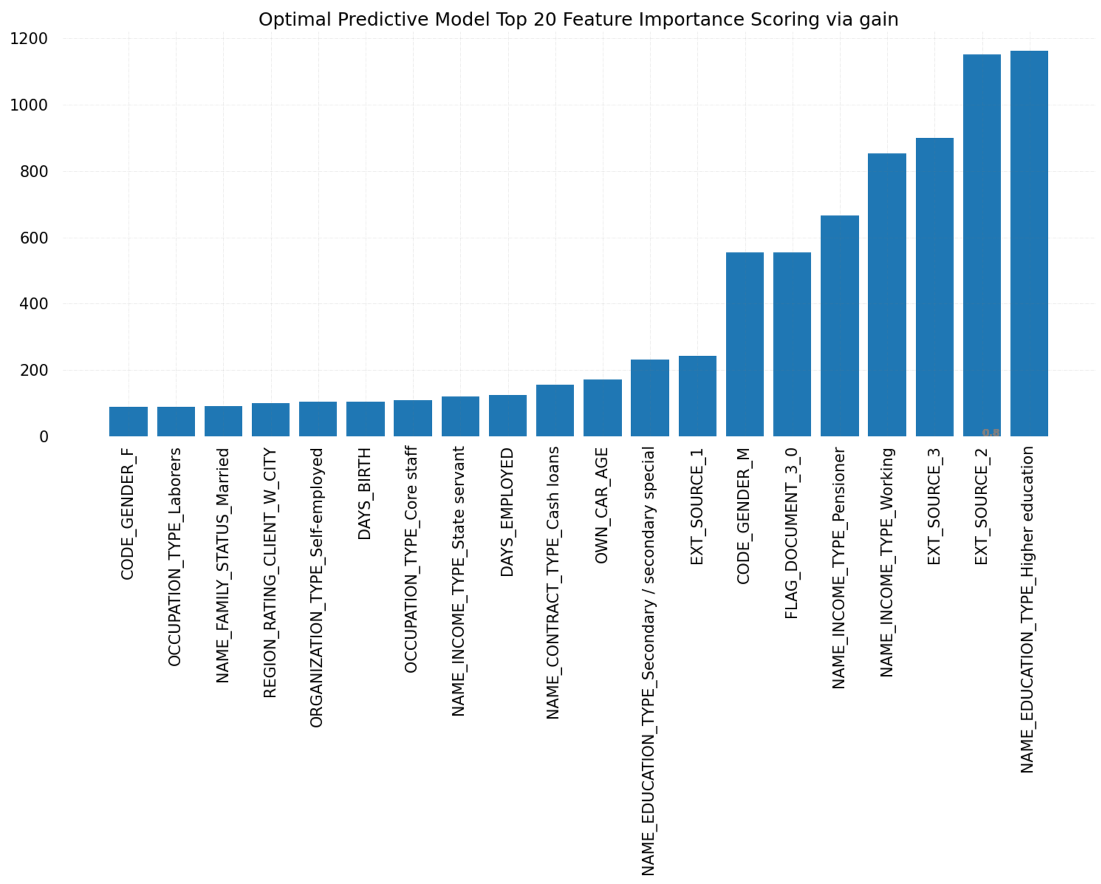

Figure 7: Feature Importance of Optimal XGB Model by Gain

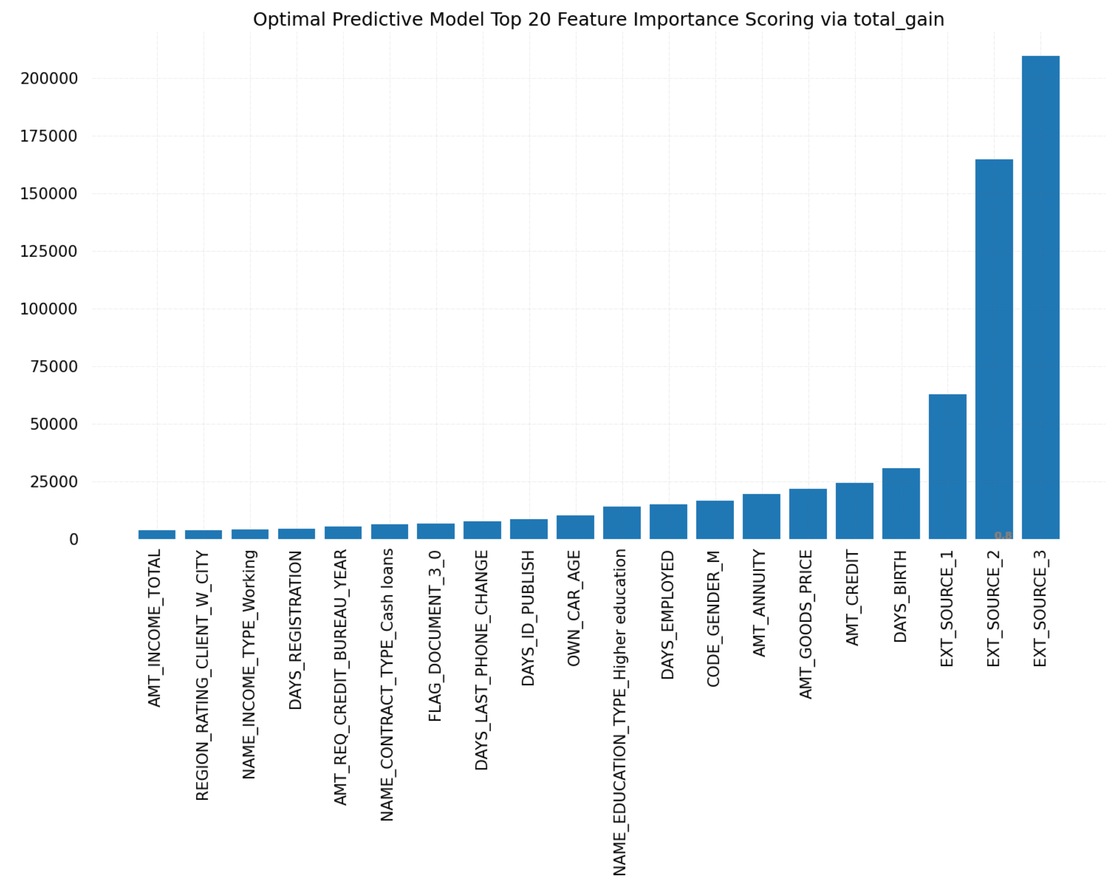

Figure 8: Feature Importance of Optimal XGB Model by Total Gain

When it comes to the cover and total cover score of the model, we only applied the total cover scores since the distinguished cover scores are hard to tell the difference of importance between features. The ranking plot is shown below:

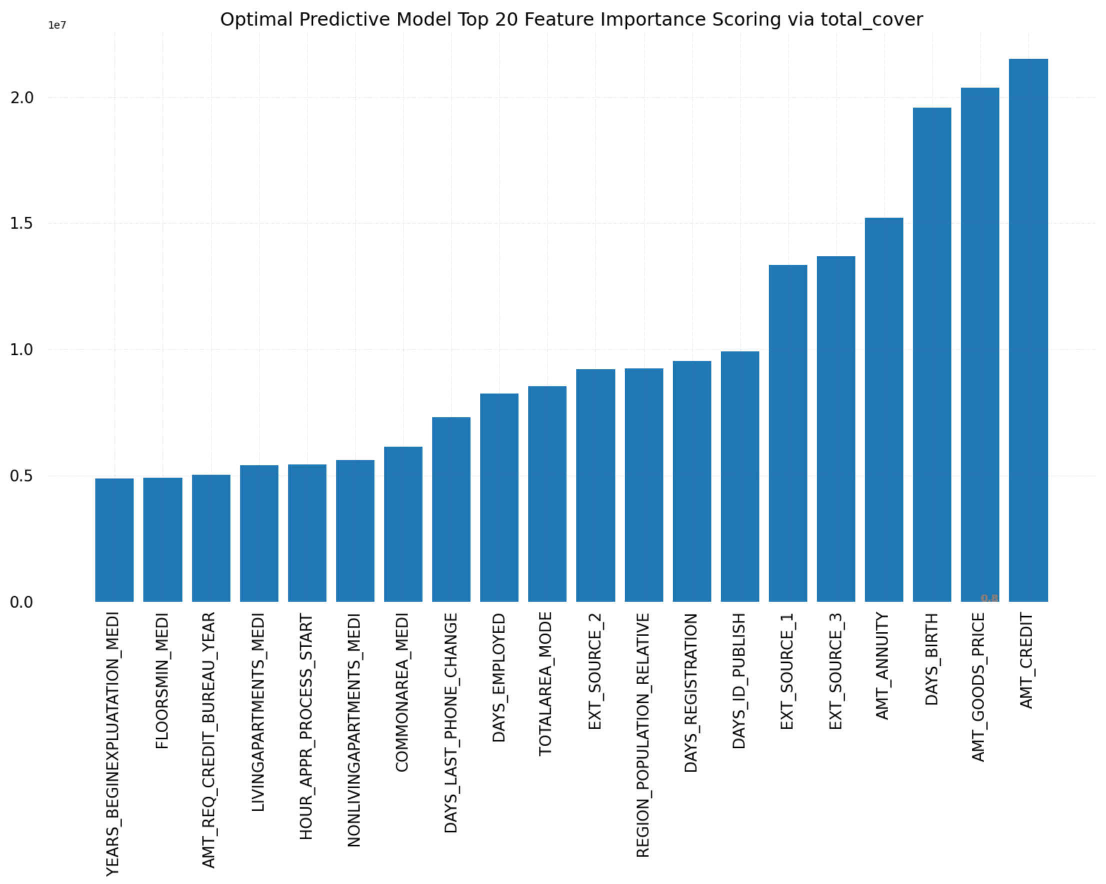

Figure 9: Feature Importance of Optimal XGB Model by Total Cover

As a conclusion, we did observe some features that were playing very significant parts in the model. For example, the credit information from other companies (noted as "EXT_SOURCE" in the plots) achieved pretty high importance scores in multiple metrics. Likewise, features "AMT_CREDIT" also received a high score since it is the credit limit – the higher, the better. Also, the number of employed days ("DAYS_EMPLOYED") is also a vital component in determining the default situation.

For the local feature importance (SHAP values), we examined the features that appeared with most of the times from the global metrics. Specifically, we plotted the dependency plot of the features and we can look at the features that are influential:

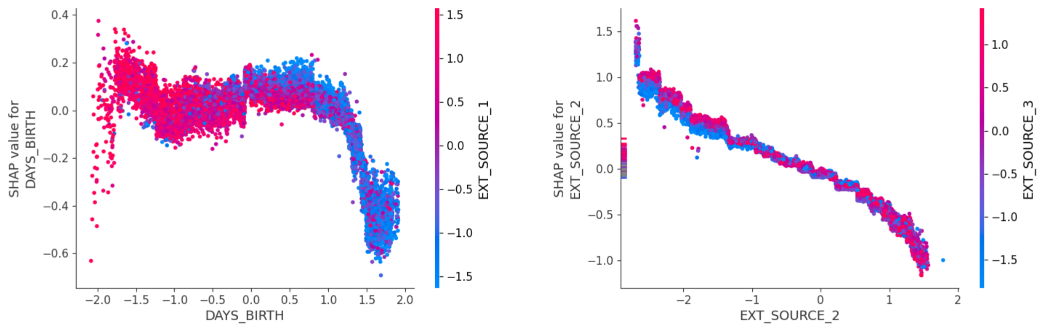

Figure 10: SHAP Dependency Plot of DAYS_BIRTH and EXT_SOUCE_2

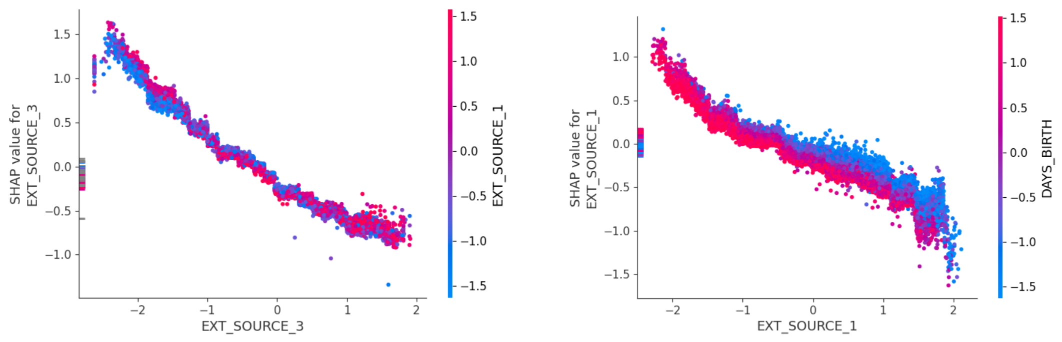

Figure 10: SHAP Dependency Plot of EXT_SOUCE_3 and EXT_SOUCE_1

The SHAP dependency plots showed us the dependencies between the features, which indicated that the "EXT_SOURCE" features are highly dependent with each other.

5. Outlook

Although in our work, the both the recall and precision are leveled up, comparing with the baseline model, the overall accuracy was compromised to achieve this. We want our optimal model not only have the outstanding accuracy score but also well-performing on identifying the minors. We suggest to try out some deep models such as neural network, to learn more complicated patterns in the dataset.

6. Reference

[1] Joe Corliss. Predicting Charge-off from Initial Listing Data, https://www.kaggle.com/pileatedperch/predicting-charge-off-from-initial-listing-data

[2] Nathan George. Some basic EDA in R and demo how to load the data, https://www.kaggle.com/wordsforthewise/eda-in-r-arggghh

[3] Anuradha. Credit EDA Study https://www.kaggle.com/anuradhamohanty/credit-eda-study

[4] Luke. Demonstrating Corrupted/Mal-formed Rows, https://www.kaggle.com/lukemerrick/demonstrating-corrupted-mal-formed-rows

[5] Tsai, Chih-Fong. Chen, Ming-Lun. (2010). Credit Rating by Hybrid Machine Learning Techniques. Applied Soft Computing. 10 (2010) 374-380.

[6] Khandani, A. E., Kim, A. J., Lo, A. W. (2010). Consumer credit-risk models via machine-learning algorithms. Journal of Banking Finance, 34(11), 2767-2787.

[7] N. V. Chawla, K. W. Bowyer, L. O.Hall, W. P. Kegelmeyer, "SMOTE: synthetic minority over-sampling technique," Journal of artificial intelligence research, 321-357, 2002.

[8] Breiman, “Random Forests”, Machine Learning, 45(1), 5-32, 2001.

[9] Tianqi Chen, Carlos Guestrin, “XGBoost: A Scalable Tree Boosting System”