Default Risk Prediction Models

Final Report

Grace Lam, Ariane Yu, Sally Yu

1 Introduction

For many people with low or no credit histories, applying for loans can be a struggle. In today's society, most banks and finance institutions largely rely on credit scores to determine a person's economic mobility and financial stability. For the unbanked population, this becomes a big barrier. The advances in machine learning provide an opportunity to utilize algorithms that use non-credit factors to determine a person's ability to repay a loan. This will provide more accessible loans for the underserved population and promote financial inclusion. By leveraging machine learning, underlying trends, and patterns can be found to make accurate predictions that may not be apparent otherwise. In this project, we hope to build inclusive lending models that can use numerous non-credit factors such as education level, income, and occupation to correctly predict creditworthiness.

2 Dataset

The main dataset application_train.csv provided by the challenge contains 300,000 loan applications and 120 features in total. Some sample features include age, income, number of family members, and occupation. From this dataset, we randomly selected 100,000 rows to use in our project. Out of the 100,000 data points, we split the dataset into a training set (80%) and a testing set (20%), and set aside the test set to use as our final evaluation of our models. Besides the main file, the challenge also provided 6 other files containing information about POS_CASH_balance, bureau, bureau_balance, credit_card_balance, installments_payments, and previous$_{a}pplication.$

- **POS_CASH_balance:** Monthly balance snapshots of previous POS (point of sales) and cash loans that the applicant had with Home Credit.
- **Bureau:** All client's previous credits provided by other financial institutions that were reported to Credit Bureau
- **Bureau_balance:** Monthly balances of previous credits in Credit Bureau.
- **Credit_card_balance:**one row for each month of history of every previous credit in Home Credit (consumer credit and cash loans) related to loans in our sample.
- **Installments_payments:** Repayment history for the previously disbursed credits in Home Credit related to the loans in our sample. One row for every payment that was made plus one row each for missed payments.
- **Previous_application:** All previous applications for Home Credit loans of clients who have loans in our sample. One row for each previous application related to loans in our data sample.

In order to take full advantage of all potential features, we read the HomeCredit_columns_descirption.csv, the variable dictionary, and performed feature engineering along with a series of merges to combine all datasets with application_train.csv. Based on both the dictionary and manually scanning their value counts, we tried to understand the data before splitting them into the training and testing sets. The detailed process will be explained further in the Feature Engineering section.

3 EDA Findings

After a first scan of our merged dataset's statistics, we found the target class (default vs. non-default) to be noticeably unbalanced where the default class vs. not default is 1 to 11.5. This is reasonable as the records that banks hold on individuals whom they have lent to are already those selected by the banks after careful evaluation of their credit worthiness. Therefore, there should only be a small proportion of clients who have defaulted after being given a loan. Although unbalanced data can cause issues in model building as bias may be introduced and can hugely affect the class weight in predictions, we chose not to balance the training data and retain the original class ratio since in the real world, the majority of loan applicants are able to pay back.

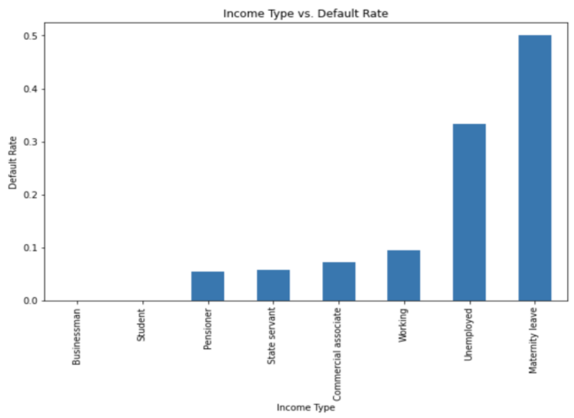
As we explored the final merged dataset that contains a total of 353 features, we scanned the HomeCredit_Columns_description.csv to understand what each column of data represents and made graphs to help visualize the relationships. Some features were found to

[0]dataset description:
https://www.kaggle.com/competitions/home-credit-default-risk

be especially correlated with the target variable. For categorical features, we made bar plots and heatmaps. For numerical features, we used boxplots, histograms, and density plots.

A few examples are provided below.

-

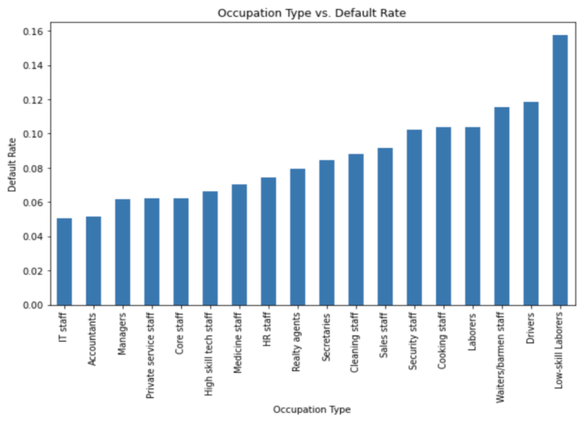


1

Figure 1: income Type vs. Default rate

Income_Type has an apparent correlation with the default rate. As shown in the bar plot, individuals on maternity leave and those that are unemployed are much more likely to default than others.

-



1

Figure 2: Occupation Type vs. Default rate

Occupation_Type is also shown to be correlated with the default rate to some degree. The more high-tech a person's occupation is, the less likely they are to default.

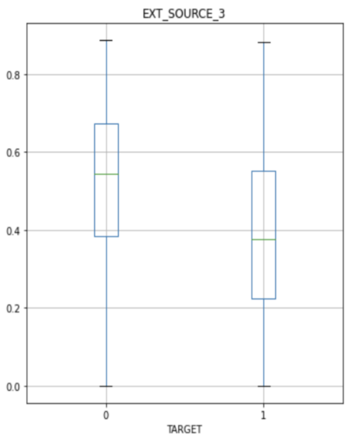


Figure 3: box plot of EXT_SOURCE_3

EXT_SOURCE_3 is a normalized score from an external data source. As shown in the box plot above, the default class and non-default class differ in the mean of the distribution of Ext_source_3 values by approximately 0.2 with relatively similar variance.

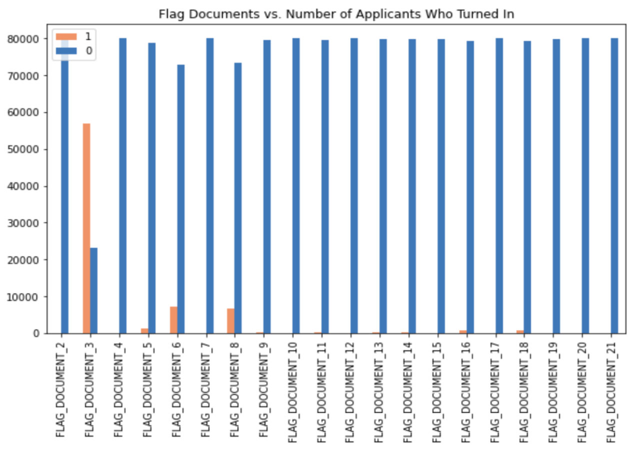


Figure 4: Flag Documents vs. Number of Applicants Who Turned In

FLAG_DOCUMENTS: there are a total of 20 flag document columns, each indicating one type of document. In the stacked bar chart above, there are two bars for each document with height representing the number of applicants who turned in that document. Since most of the documents have very few people who submitted, we later combined the 20 columns by summing up the number of documents each applicant turned in and dropped the flag_documents columns with less than 10 occurrences.

4 Feature Engineering

In the first round of feature selection and feature construction, we started by merging bureau.csv and bureau_balance.csv on the unique 'SK_ID_BUREAU'. Bureau_balance is a time series data with continuous updates of the client status. Therefore, we took the most recent update of each existing ID and engineered a new feature of how many times the status has been updated. Upon merging with bureau, we first one-hot-encoded the text data that existed in bureau to preserve text data from later merging on 'SK_ID_CURR'. For each variable in the merged bureau dataframe, we grouped by 'SK_ID_CURR', took their max and sum, and engineered a new numeric feature that represents the proportion of active accounts that each new applicant owns out of all the accounts. We ended up engineering the condensed bureau dataset by merging it into the application dataframe, creating an intermediate dataframe with the dimension (100000, 205).

POS_CASH_balance, credit_card_balance, installments_payments share the same 'SK_ID_PREV' as in the previous_application. POS_CASH_balance contains both numeric and text data, and we found that the null text data was inconsistently represented throughout the data sets with different string content or NaN. Therefore, we chose to replace all null text values with the string "XNA" to maintain data consistency. After filling in text data, we performed one-hot-encoding to preserve information in this time-series dataset. Before grouping the frame by 'SK_ID_PREV', we created two new variables. One is the count of past paid installments and the another one is the proportion of the number of past paid installments to the number of installments that the individual needed to pay. Then, we grouped the set by 'SK_ID_PREV', took the most recent update of the account, and constructed a new variable as to how long the account had been active. Along with grouping the accounts owned by the applicants to retrieve the average numeric data of each applicant, we counted the number of accounts that each ID owned.

The original previous application contains 37 columns. As we read through the documentation on the features and tried to understand their implication in evaluating the credibility of the applicants, we eliminated some irrelevant as well potential confounding variables, saving only 17 features in total. We then one-hot-encoded the text data and constructed a new feature as if the applicant is missing the down payment. To avoid later handling NaN would impact non-existing data in the previous application dataframe, we immediately filled the null values with 0. Looking at the credit_card_balance and its features, we selected some numeric features and took their mean and the max days past due by 'SK_ID_PREV'. Along with one-hot-encoded text data, we merged all the engineered credit_card_balance dataframe with the previous application.

For installments_payments, we first made a new variable on the differences between the days when the installment was paid and the days when the installment was supposed to be paid. Then, we grouped by the 'SK_ID_PREV' and aggregated the mean of the day differences and the sum of the actual payments and supposed payments. After aggregating, we constructed a new feature on the proportion of the actual payment amounts to the required payment amounts. At the last step of merging, we created a full application dataframe with a dimension of (100000, 354).

In the condensed dataframe where all 7 csv files were merged, we filled leftover null text data from the original application with 'XNA' as before and one-hot-encoded them. Along with a few newly created features, we finalized our dataframe with a total of 485 features. After splitting the data into training and testing sets, we individually utilized different imputing strategies on applicants who missed available past data, such as mean imputer and KNNimputer.

5 Evaluation metric

False positives and false negatives are 2 types of errors that exist in binary classifications. In our dataset, false positive refers to the case where the model predicts that an applicant will default, while in reality, they will pay off the loan. False positives can be costly and should be avoided since the banks would miss out on potential profits and applicants will lose opportunities. On the other hand, the false negatives in loan default prediction are when an applicant is predicted to be able to repay a loan but ends up defaulting on it. It is also important to minimize this error since it will cause significant financial loss to banks.

Since both false positives and false negatives have bad implications, we want to minimize both types of errors. To do so, we decided to use the roc_auc score as the measure of our model's performance. roc_auc is an evaluation metric that takes both the true positive rate and false positive rates into account and produces a combined score. This characteristic makes it a good metric for unbalanced data such as the loan default population.

6 Baseline models

Baseline models are trained on the main file, application_train.csv. A ColumnTransformer was used to perform mean imputation on missing values and One-hot encoded the categorical features.

6.1 Logistic Regression:

Logistic regression is a useful model when dealing with binary classification. It is good at modeling relationships within data, computationally efficient, and can handle large datasets. We started with a logistic regression model with no custom parameters, which resulted in a roc_auc score of 0.5 and predicted all instances to be class

0s (will not default). This is likely due to the markedly imbalanced dataset causing the model to be biased towards the majority class. Thus we decided to add class weights to account for this. After adding a class weight parameter of 0:1, 1:11.5 the model was able to achieve a score of 0.574.

6.2 Random Forest Classifier:

Random forest is an ensemble method that is based on decision trees, it combines the predictions made by a collection of decision trees trained on different subsets of the data. For a random forest, it is important to control the complexity of the model by adjusting parameters such as max depth to avoid overfitting. Therefore, in addition to adding class weights, we limited the max_depth parameter to 15. The model was able to get a roc_auc of 0.625.

6.3 Baseline result:

The best score achieved for the baseline models was 0.625. The best-performing model was a random forest classifier with class weight and max depth parameters. From performing baseline models, we realized that conducting hyperparameter tuning and specifying class weight in our model training is critical.

7 Model

7.1 LinearSVC:

Linear Support Vector Classifier constructs hyperplanes that separate data into different classes. This algorithm is used instead of regular SVC because non-linear SVC is more computationally costly for our large dataset. With hyperparameters (C = 0.01, class_weight={0:1,1:11}, tol=0.001, max_iter=800), the model achieved a score of around 0.697 using the normalized untrimmed dataset.

7.2 Logistic regression:

We used hyperparameters, (penalty='l1', solver='saga', class_weight={0:1,1:11}), to achieve a score of around 0.695 by training it with normalized data. In testing this model, we also conducted PCA decomposition and reduced features to only principal components that explained more than 0.98 variance. However, the classification efficacy was largely reduced to around 0.53, suggesting that dimensionality reduction may not be ideal to apply to this particular dataset.

7.3 Random Forest Classifier:

After a round of grid search, the best hyper-parameters were found to be (max_depth = 7, n_estimators=1100, class_weight={0:1,1:11}), which achieved a test score of 0.671. We also selected features based on its outputted feature importances. If we selected 300 out of 480+ features, the scores generally fluctuated about 0.002 around the scores of models fed by the complete training set. The insignificance suggests that this feature selection strategy also may not be very optimal for this dataset.

7.4 Lightgbm:

The data used to train this model includes the 350 best features selected by RFE (recursive feature elimination) with lightGBM as the estimator. The categorical features were one-hot encoded with certain categories eliminated due to the low frequency of occurrence. The model optimization was done by performing grid search on hyperparameters such as learning_rate, num_leaves, and max_depth. The final model has the parameters: (max_depth=10, learning_rate=0.06, n_estimators=200, num_leaves=15, class_weight={0:0.08,1:0.92}), and the score on the validation set was 0.698. Since It achieves one of the highest performances, we decided to include it in the final stack.

7.5 XGBoost:

XGBoost Classifier is known to be good at handling large and high-dimensional data and is relatively fast to train. However, as one of the stronger models, it is prone to overfitting. To avoid this and to optimize on its generalization performance, we performed grid search on n_estimators, max_depth, subsample, and reg_alpha. The final hyperparameter values that gave us a best score of 0.696 were (max_depth = 2, n_estimators= 1000, reg_alpha = 10, subsample= 0.6,learning_rate=0.05)

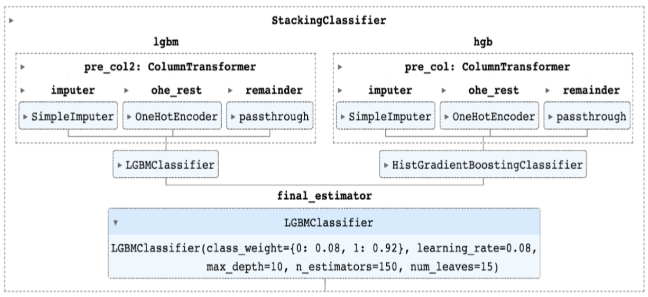
7.6 Hist Gradient Boosting Classifier:

Histogram-based Gradient Boosting Classification Tree is a histogram-based boosting algorithm of Decision Tree. It is robust to outliers and null values because of its method of binning and way of handling missing values in features. The fine-tuned model of HistGraidentBoosting Classifier could achieve a score of around 0.694 with hyper parameters (learning_rate=0.08, max_iter=800, max_depth=3, scoring='roc_auc', interaction_cst='pairwise', tol=0.0003, class_weight={0:1,1:11}, l2_regularization=5).

7.7 Final Model:

To optimize our models and improve our score, we spent significant time on feature selection and model selection. In feature selection, we tried multiple approaches including Univariate Feature Selection, Lasso Regression Forward Selection, Random Forest, PCA decomposition, and RFE. Most proved to have similar performances but a consistent pattern was found, which was that relatively more features resulted in better overall performance. As a team of three, we split up the task of exploring different imputation techniques and types of models. After many rounds of fine-tuning and comparing, the best models were found to be Logistic Regression Classifier, XGBoost Classifier, and lightGBM.

In order to take full advantage of these models, we stacked together the predictions made by each of these models both manually by taking the majority vote and through using a stack. However, the stacks made of multiple different best-performing models did not achieve higher scores than the individual models, some even had much lower scores. After rounds of comparison, we found the best-performing stack to be the one consisting of only a lightGBM Classifier and a HistGradientBoosting Classifier.



```
LGBMClassifier(class_weight={0: 0.08, 1: 0.92}, learning_rate=0.08,
               max_depth=10, n_estimators=150, num_leaves=15)
```

Figure 5: Final model pipeline

7.8 Results:

Our final model, which consists of a stack of lightGBM Classifier and a HistGradientBoosting Classifier, achieved a final score of 0.7011 on the test set.

8 Literature

Our project used datasets provided by the Home Credit Default Risk Kaggle Challenge. The challenge aimed to use alternative data and machine learning methods to predict an applicant's repayment abilities, which will help broaden financial inclusion for the unbanked population who have little to no credit histories. By building a model that uses various factors other than credit score to predict a person's financial stability, more people will have access to loans and loans will be given with a calculated principal, maturity, and repayment calendar that empower the clients.

The top-performing teams mostly focused on feature engineering and feature selection. With some background knowledge and research into credit scoring, many teams were able to construct a considerable number of useful features on top of the ones provided. They then performed a series of forward feature selection processes using different techniques to reduce dimensionality and filter out non-important features. These teams also used a variety of different base models, including Linear Regression, LightGBM, XGB, and neural network. In the end, the final model was a stack of best-performing base models, which achieved very high scores, averaging around 0.8 roc_auc. In retrospect, they tested the performance of each of their individual base models on the final test set, which turned out to be top-performing as well. This shows that having strong base models in a stack is critical to their success.

From the experiences of the winning team, we learned to focus on feature engineering. In addition, we wanted to also spend time on trying various models that were not used by the winning teams. Namely, we tried Naive Bayes, Adaboost, Bagging Classifier, Extra Trees Classifier, Hist Gradient Boosting Classifier, linearSVC on top of Linear Regression, XGBoost, and lightGBM. By trying a variety of models, we were hoping that they would capture different trends in the data so that an average of several models can achieve a higher score than each individual model. By trying these simpler models, we also wanted to test if the underlying relationship between most features and default rate can be captured linearly without the use of computationally-expensive models like Neural Network. During the process, many models were eliminated as their individual performances were much lower than optimal, but some models did perform better than expected, such as linearSVM and Linear Regression, which we kept until building the final stack.

9 Comparison

9.1 The best performing teams:

The best-performing teams all had members who have some domain knowledge in loan defaults and constructed their models upon their past experiences in areas such as credit scoring. From applying this knowledge, these teams were able to engineer and generate a significant amount of useful features to be incorporated into their models. In addition, they used different combinations of features and subsets of data when training each of their individual models. By exploring all kinds of models, they were able to select a variety of different models with distinct advantages in predicting defaults and eventually combine them into a stack. Doing so, the final stack is able to average over different errors made by each model to achieve an overall high score. Upon reflection, some of the most successful models include Logistic Regression, lightGBM, and Neural Network.

9.2 Our model:

For our models, we created new features using basic aggregations during the merging step, which gave us a total of 353 features before encoding categorical features. But due to a lack of domain knowledge in loan defaults, the features we created may not all be very predictive. Therefore, many of the features we used in training our models do not actually contribute to any boost in performance. In order to make up for the lack of useful features, we focused on feature selection and model selection to make sure our models are optimized to the best degree possible. Therefore, we spent a considerable amount of time on hyperparameter tuning using grid search. Since we explored many different types of models, we were able to stack together the best-performing models into a final model that gave us a final boost in performance.

9.3 Performance:

The best performance in the Kaggle competition was 0.806, and the performance of our final model is 0.702. We realized that there is definitely a gap between our model and the winning team's. Besides our lack of experience with building machine models, we also lack high computing power and a certain degree of domain knowledge in the subject matter. This prevented us from improving our models to scores beyond 0.7.

Aside from our lack of domain knowledge in credit evaluation that prevented us from constructing more valuable features, we had limited computational resources. For example, we tried to stack more models utilizing the Stacking Classifier in sklearn, but the model ran extremely slow and we were unable to tune it. We also tried to use the entire data sets provided by the challenge. Because of the high dimensionality of this data, using more training examples would definitely boost the classification result. However, trying to merge the data or even loading the entire dataset was challenging for some of our members. The large size of the dataset caused our kernel to run out of memory and crash several times, which prevented us from running complex algorithms. Although computational power was a limiting factor, we realized that our inexperience in building machine learning models could hinder our model selection as well as model optimization processes.

10 Takeaways:

- Feature engineering is critical in improving model performances. In order to effectively construct useful new features, understanding the underlying data generation process is hugely important. This often requires the use of certain domain knowledge, which we don't have. If given more time, we would conduct more research on credit scoring in order to generate unique and effective features to add to our feature set.

- Step-by-step feature selection is important although it is time intensive. Techniques such as forward selection and backward elimination are a comprehensive way of evaluating each feature's predictive power. Because of our inexperience in feature selection, we only realized its importance in the later stage of our project. Due to the limited computational power of our devices and time constraint, we were unable to utilize these exhaustive methods to our advantage, which may have prevented further improvement of our models.

References

[1] Kaggle dataset: https://www.kaggle.com/competitions/home-credit default-risk

[2] Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011. Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., . . . Liu, T.-Y. (2017).

[3] Lightgbm: A highly efficient gradient boosting decision tree. Advances in Neural Information Processing Systems, 30, 3146–3154.

[4] 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). New York, NY, USA: ACM. https://doi.org/10.1145/2939672.2939785

9