Machine Learning Engineer Nanodegree
Capstone Report

Credit Scoring

by Deviyanti Aryani Mariam
November 2021

I. DEFINITION

Domain Background

As technology is evolving so fast that it is getting to the point where it is able to create a new digital revolution in the finance sector by providing access to more people, lending companies face a challenge in identifying customers' creditworthiness, the ability of the customers to pay back. This project is meant to help company "X" become more efficient and effective in making decisions and providing financial services to its customers. The credit scoring model will be used as a tool for accepting or rejecting a loan application and it will be also for determining the cutoff of the credit score as the company tries to satisfy their expected approval rate (the number of loans will be approved out of the number of applications) and default rate (the number of non-performing loans out of the number of approved loans). Machine learning would be the approach used for the credit scoring. The model defines customer scoring based on past borrowers' characteristics.

Problem and Solution Statement

The upper management of company "X" wants the overall default rate of their portfolio to be below 3% while the approval rate is not less than 70%. Below is the main objective of the project:

- Build a model that predicts whether a loan would be default or not, the result would be the predicted probability of default which indicates a customer is unlikely to pay
- Provide recommendations on the optimal credit score cutoff.

Model development could be solved by using supervised machine learning algorithms like Logistic Regression and XGBoost. We would do visualizations and feature engineering. As it takes a lot of iterative work, we try to select multiple feature combinations to be fed into the algorithms and do hyperparameter tuning and choose the model with the best evaluation metric value and the model should satisfy the requirements from management. We also will provide the credit scorecard that contains information about score bins, their approval rate and default rate. The scorecard enables the management to accept a certain number of loans based on their willingness to take on risks.

Datasets and Inputs

Dataset can be accessed at
https://github.com/deviyantiam/credit_scoring/blob/main/data.csv
It is artificially created based on real data to replace privacy sensitive information.

Feature	Description
x	user_id
number_of_cards	number of cards owned by the customer
outstanding	total outstanding amount of credit card usage
credit_limit	credit limit amount that can be used
bill	last month customer bill amount
total_cash_usage	last month total cash usage of the customer
total_retail_usage	last month total retail usage of the customer
remaining_bill	remaining bill that has not been paid in the last month
branch_code	branch code

payment_ratio	payment per bill ratio in the last month
overlimit_percentage	overlimit percentage
payment_ratio_3month	payment per bill ratio in the last 3 month
payment_ratio_6month	payment per bill ratio in the last 6 month
delinquency_score	delinquency score
years_since_card_issuing	total year since first card issued
total_usage	total usage
remaining_bill_per_number_of_cards	ratio remaining bill per number of cards
remaining_bill_per_limit	ratio remaining bill per credit limit
total_usage_per_limit	ratio total usage per limit
total_3mo_usage_per_limit	ratio total 3 months usage per limit
total_6mo_usage_per_limit	ratio total 3 months usage per limit
utilization_3month	Credit card utilization for past 3 months
utilization_6month	Credit card utilization for past 6 months
default_flag (LABEL)	Credit default flag (1: default; 0: non_default)

Evaluation Metrics

The model performance will be graded based on the area-under-the-ROC-curve score (AUC) or GINI (2*AUC-1). The Receiving Operating Characteristic (ROC) curve is a graphical plot of the True Positive Rate (percentage of bad rejected) versus the False Positive Rate (percentage of goods rejected) for every threshold or cut-off (see **Figure 1**). The Area under the Curve (AUC) measures the percentage of the area that is under this curve. The higher the percentage, the better the model performance. Gini is a normalised or linear transformation of AUC, a random classifier scores 0, and a perfect classifier scores 1. As the data is imbalanced, accuracy is not the best metric to use to qualify our model. Therefore we use ROC-AUC because of its sensitivity towards True Positive and False Positive.

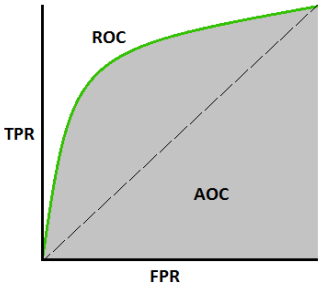


Figure 1 ROC-AUC

II. ANALYSIS

Data Exploration

The data has 15645 rows and 24 columns in total. Data contains:

- User_ID or the column named X.
- 21 numeric columns, as follows:
    - Number_of_cards
    - Outstanding
    - Credit_limit
    - Bill
    - Total_cash_usage
    - Total_retail_usage
    - Remaining_bill
    - Payment_ratio
    - Overlimit_percentage
    - Payment_ratio_3month
    - Payment_ratio_6month
    - Delinquency_score
    - Years_since_card_issuing
    - Total_usage
    - Remaining_bill_per_number_of_cards
    - Remaining_bill_per_limit
    - Total_usage_per_limit
    - Total_3mo_usage_per_limit
    - Total_6mo_usage_per_limit
    - Utilization_3month
    - Utilization_6month
- 1 categorical column
    - branch_code

The data is imbalanced with only 9% positive (customers tend to have no ability to pay back) and it has missing values, as detailed below

Table 1 Missing Value Ratio

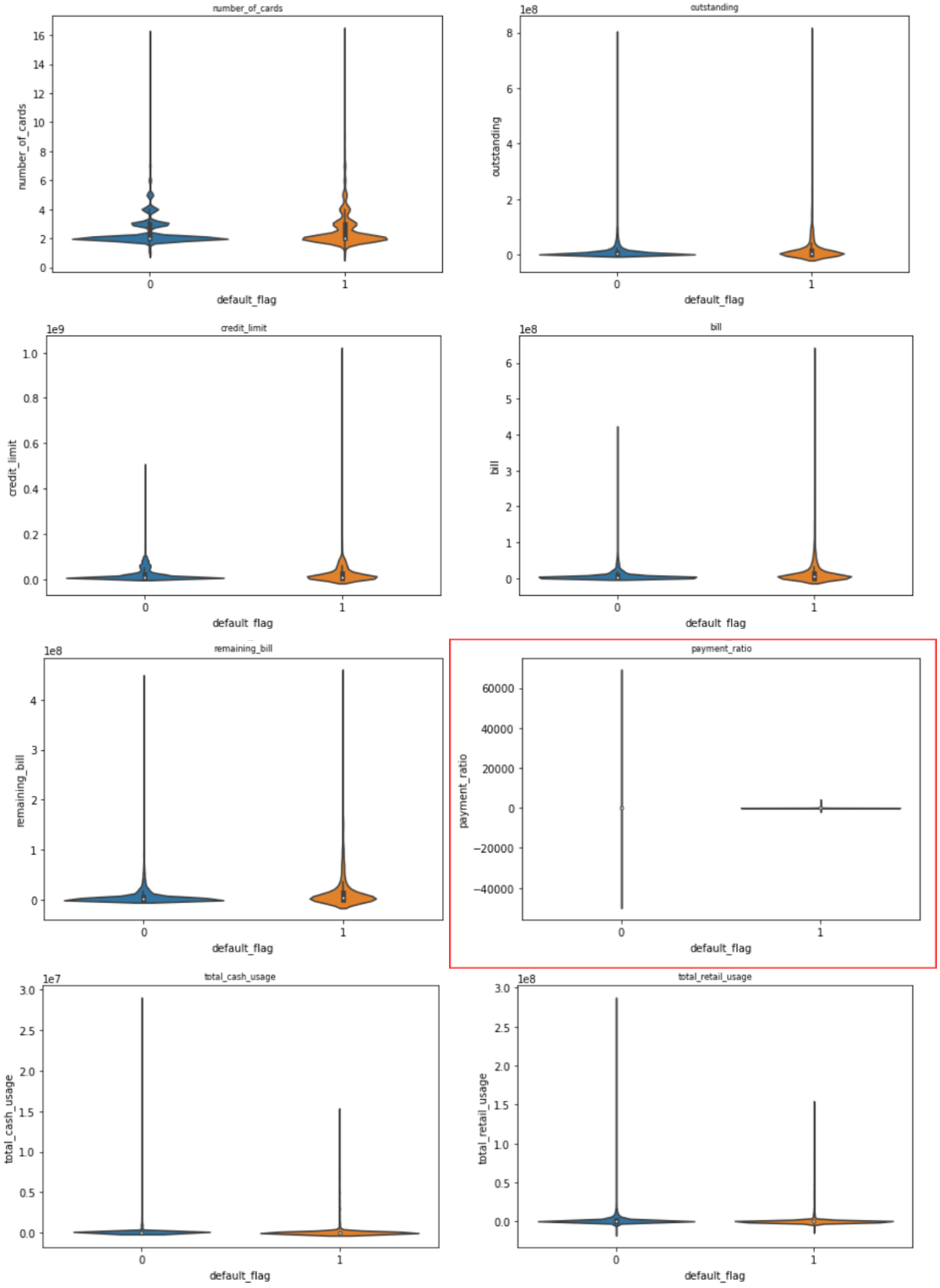
column	total_row_missing	missing_value_rate
total_cash_usage	45	0.29%
branch_code	195	1.25%
overlimit_percentage	26	0.17%
delinquency_score	88	0.56%
utilization_6month	2842	18.17%

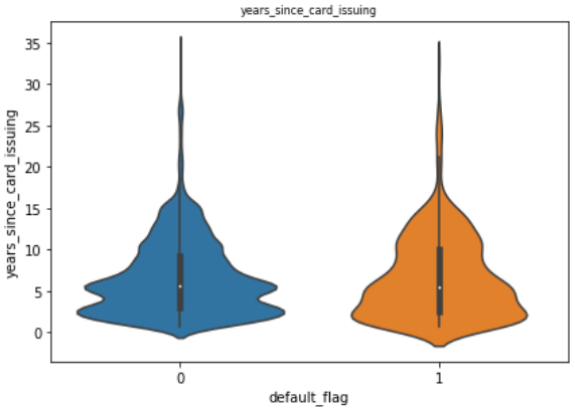
As there is no column with more than 50% missing value rate, no column would be discarded. We will impute missing values with their mean for numeric features and modes for categorical features as our baseline model, Logistic Regression cannot take missing values as input. Data also don't have duplicated rows. Data may have outliers, thus we will perform scaling on numerical features because our baseline model is sensitive to it.

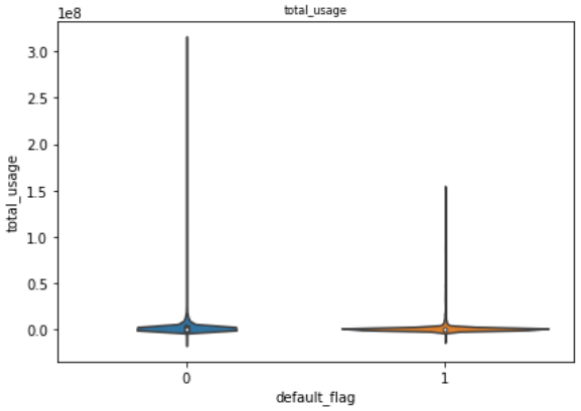
Exploratory Visualization

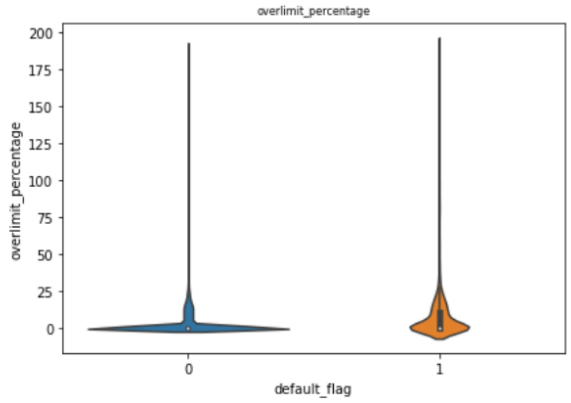
In order to build machine learning with better performance, we need to explore the datasets which includes visualizing the data distribution. In that way, we can have a better

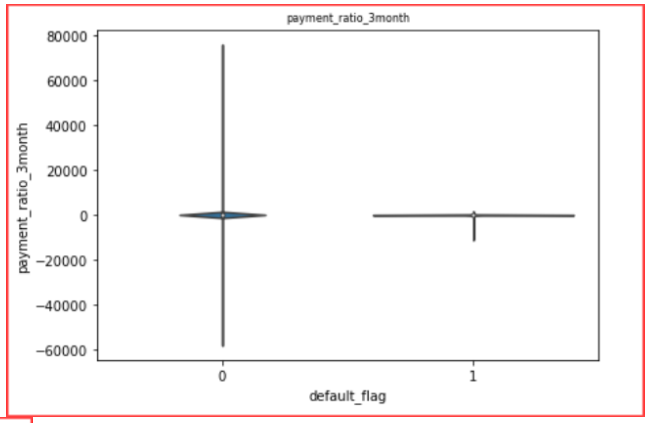
understanding of how the dataset looks like and how to select the important features to support the model implementation. Below is the distribution of each feature in the form of violin plots.

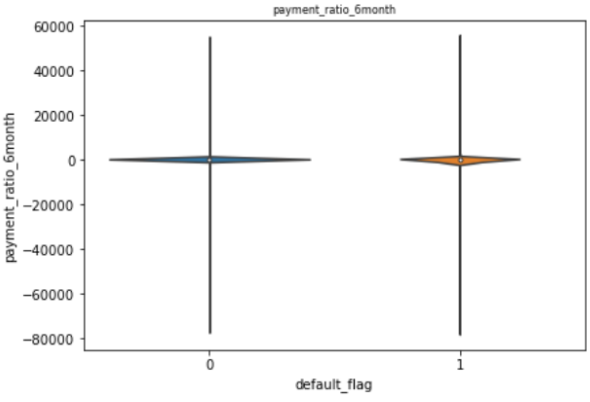


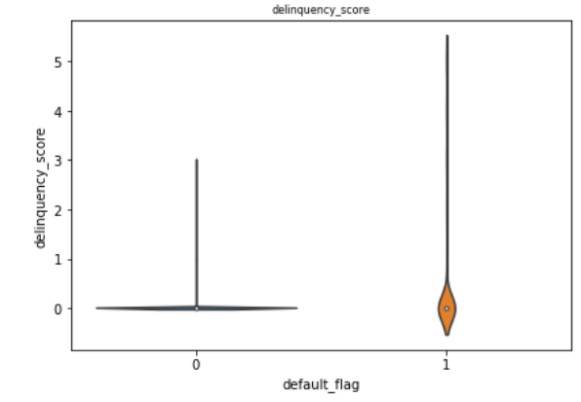

years_since_card_issuing
years_since_card_issuing
default_flag

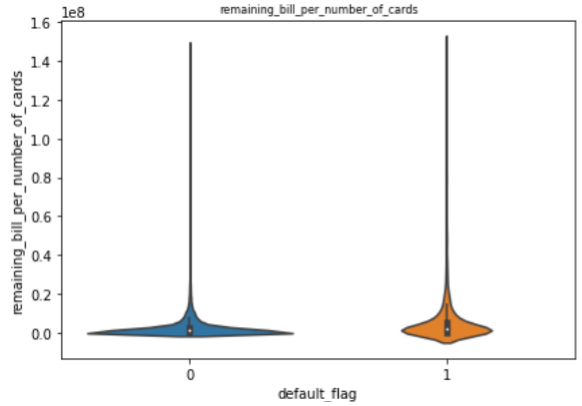

total_usage
1e8
total_usage
default_flag

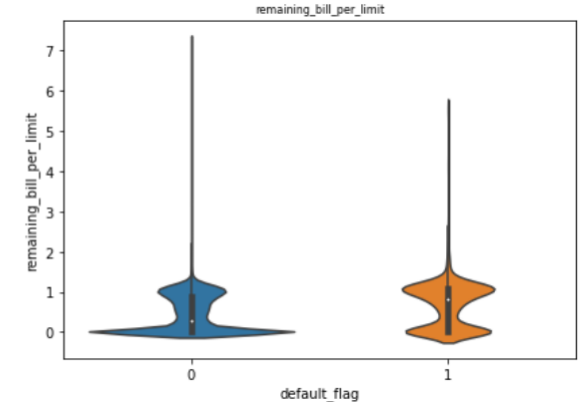

overlimit_percentage
overlimit_percentage
default_flag

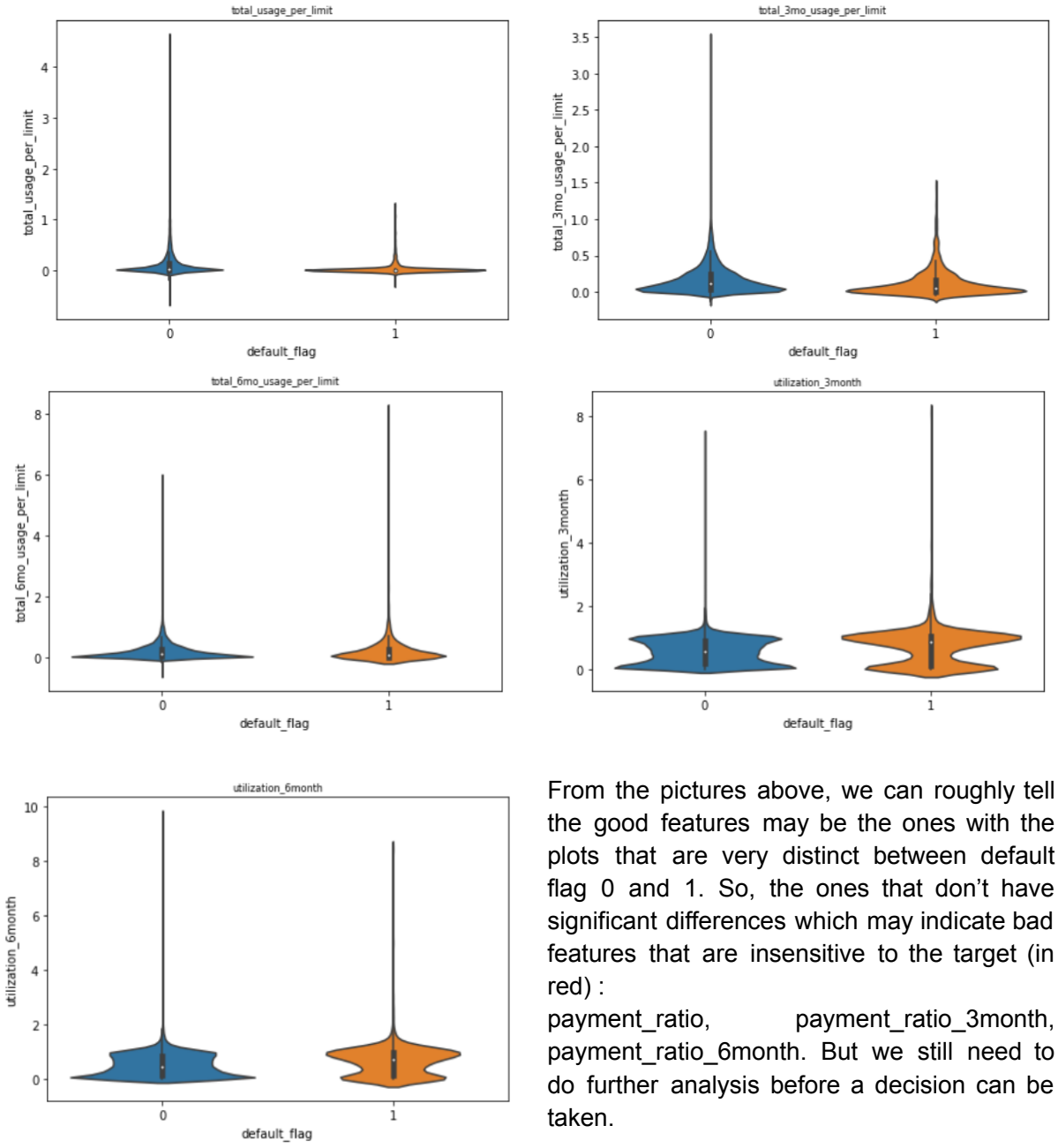

payment_ratio_3month
payment_ratio_3month
default_flag


payment_ratio_6month
payment_ratio_6month
default_flag


delinquency_score
delinquency_score
default_flag


remaining_bill_per_number_of_cards
1e8
remaining_bill_per_number_of_cards
default_flag


remaining_bill_per_limit
remaining_bill_per_limit
default_flag



From the pictures above, we can roughly tell the good features may be the ones with the plots that are very distinct between default flag 0 and 1. So, the ones that don't have significant differences which may indicate bad features that are insensitive to the target (in red) :
payment_ratio, payment_ratio_3month, payment_ratio_6month. But we still need to do further analysis before a decision can be taken.

Figure 2 Violin plot

As the plots we have seen above, they are all numerical features. We can create a visualisation for categorical features too by using a bar chart. Below will tell us that branch E is most likely to have their customer default compared to other branch codes.

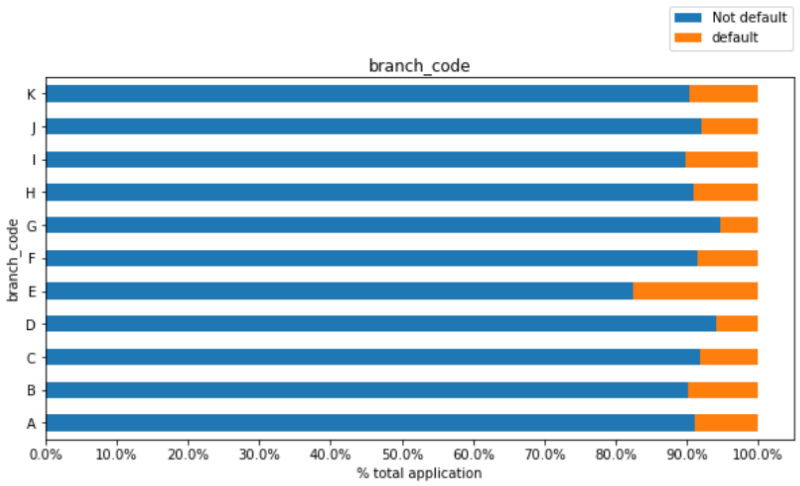


Figure 3 Bar chart of branch_code

To be more certain of the feature analysis, we can plot data in different forms of graphs or use correlation. The following is the correlation plot (go to section **Data Preprocessing** to see clearer picture)

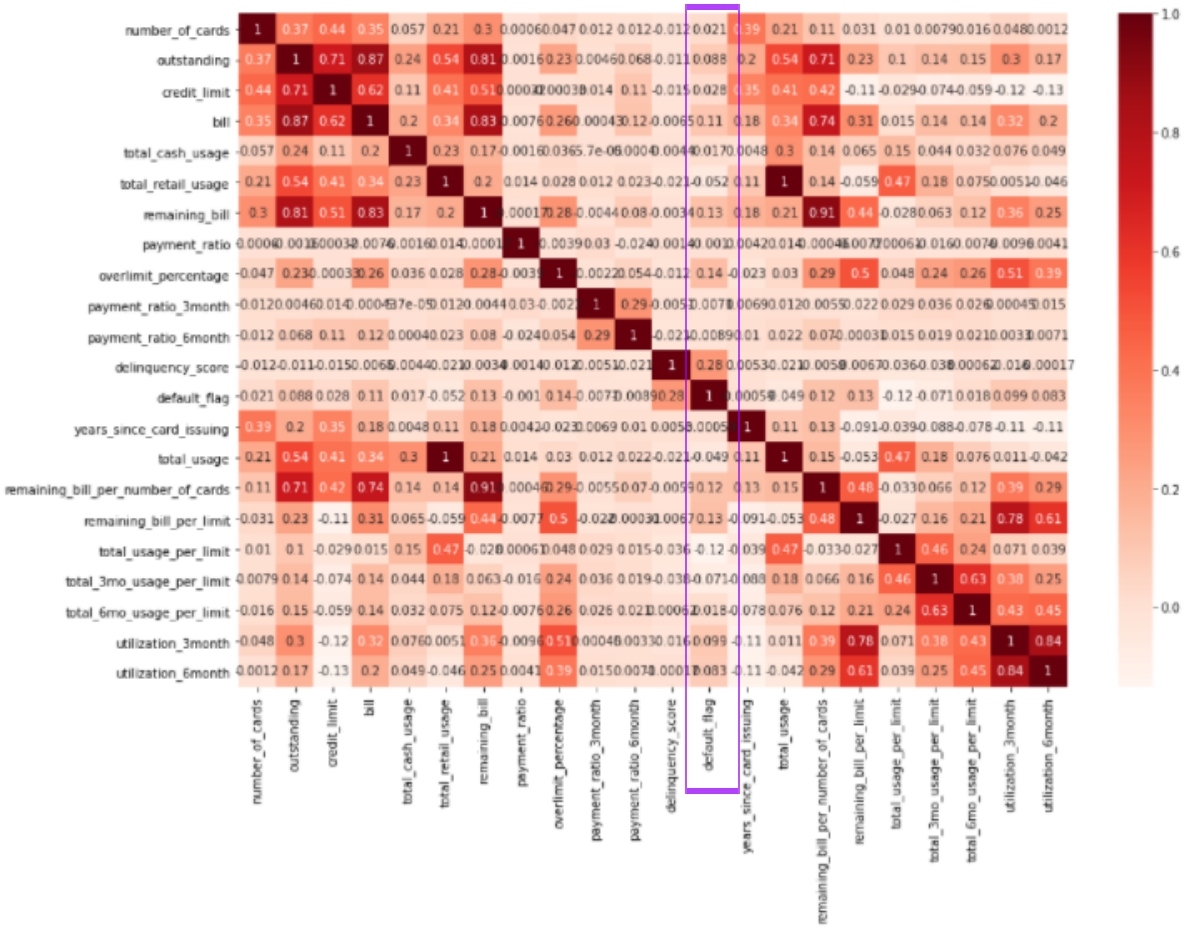


Figure 4 Correlation plot

It's interesting that some independent features may cause multicollinearity (total_retail_usage and total_usage, remaining_bill and

remaining_bill_per_number_of_cards), but don't fret much when the model we use is tree-based, the model can handle it, it would be a different story if we choose a regression model, then we need to calculate Variance Inflation Factor (VIF), and make sure the VIF values are low between the independent features. By low, it means VIF value below 10. We will show VIF values later, as we decide our baseline model is logistic regression.

Algorithms and Techniques
Below is the overview of classification algorithms used:

- **Logistic Regression**
  It is for classification tasks and not regression problems. The name 'Regression' here implies that a linear model is fit into the feature space. This algorithm applies a logistic function to a linear combination of features to predict the outcome of a categorical dependent variable based on predictor variables. Logistic Regression majorly makes predictions to handle problems that require a probability estimate as output, in the form of 0/1.
  Advantages of using Logistic Regression
  - Easier to inspect and less complex.
  - These algorithms do not assume a linear relationship between the dependent and independent variables and hence can also handle nonlinear effects.

  Drawbacks of using Logistic Regression
  - When the training data is sparse and high dimensional, in such situations a logistic model may overfit the training data.
  - It is not robust to outliers and missing values.

- **XGBoost**
  Extreme Gradient Boosting (XGBoost) is an implementation of gradient boosting machines that are highly flexible and versatile while being scalable and fast. XGBoost works with most regression, classification, and ranking problems as well as other objective functions. XGBoost is a variation of boosting - an ensemble method algorithm that tries to fit the data by using a number of "weak" models, typically decision trees. The idea is that a "weak" classifier which only performs slightly better than random guessing can be improved ("boosted") into a "stronger" one that is arbitrarily more accurate. Building on the weak learners sequentially, at every round each learner aims to reduce the bias of the whole ensemble of learners, thus the weaker learners eventually combined into a powerful model. In gradient boosting, weak learners are generalized by optimizing an arbitrary loss function using its gradient.
  Advantages of using XGBoost
  - Work with large data
  - Lots of flexibility, can optimize on different loss functions and provides several hyperparameter tuning options that make the function fit very flexible
  - Robust to missing values

  Drawbacks of using XGBoost
  - Does not perform well on unstructured data.

Benchmark Model

We built a baseline model to which the actual model's performance would be compared. We used Logistic Regression. The reason is that Logistic Regression is easy to implement. After performing data preprocessing and feature selection for this model, we got AUC 0.77

III. METHODOLOGY

Data Preprocessing

- Data type checking: if there is any feature that is not assigned to the correct type, then we need to convert it. For this case, all columns are in the correct format
- Missing Value Handling: as all missing value ratios are not big or more then 50%, then no column should be ruled out. Instead, we replace the missing values with mean for all numeric features and mode for categorical features
- Encoding: we used one hot encoding for categorical features. That is because the branch code is a nominal type, it doesn't have ordering. If the feature is an ordinal one, perhaps we can use label encoding.
- Scaling: we used StandardScaler for handling outliers as our baseline model is sensitive to it.
- Train-test splitting: test size = 20%
- We also calculate correlation and VIF to help us make decisions on feature selection. The following are the correlation between independent variables and our label and the VIF value. The higher the correlation, The stronger the relationship between the independent variable and our label. The lower VIF values are between the independent feature, The less the probability to experience multicollinearity.

Table 2 Correlation Coefficient

	default_flag
years_since_card_issuing	0.000589
payment_ratio	0.001038
payment_ratio_3month	0.00713
payment_ratio_6month	0.008944
total_cash_usage_imp	0.017121
total_cash_usage	0.017375
total_6mo_usage_per_limit	0.017959
number_of_cards	0.021001
credit_limit	0.028078
total_usage	0.049363
total_retail_usage	0.051667
utilization_6month_imp	0.067433
total_3mo_usage_per_limit	0.070596
utilization_6month	0.082939
outstanding	0.08764
utilization_3month	0.099121
bill	0.105897

total_usage_per_limit	0.118604
remaining_bill_per_number_of_cards	0.120161
remaining_bill	0.12749
remaining_bill_per_limit	0.134803
overlimit_percentage_imp	0.139366
overlimit_percentage	0.140546
delinquency_score_imp	0.275663
delinquency_score	0.275741
default_flag	1

Table 3 VIF

variables	VIF
number_of_cards_sd	1.710914
outstanding_sd	10.268372
credit_limit_sd	3.15232
bill_sd	5.484028
total_cash_usage_sd	3378.933001
total_retail_usage_sd	553488.8754
remaining_bill_sd	12.246902
payment_ratio_sd	1.003176
overlimit_percentage_sd	1.452046
payment_ratio_3month_sd	1.099773
payment_ratio_6month_sd	1.133222
delinquency_score_sd	1.024386
years_since_card_issuing_sd	1.281313
total_usage_sd	576557.4381
remaining_bill_per_number_of_cards_sd	7.99741
remaining_bill_per_limit_sd	3.714933
total_usage_per_limit_sd	1.727658
total_3mo_usage_per_limit_sd	2.25695
total_6mo_usage_per_limit_sd	1.910868
utilization_3month_sd	5.990281
utilization_6month_sd	2.521177

VIF higher than 10 indicates high multicollinearity between this independent variable and the others. So for building our baseline model, the following features would be used: Number_of_cards_sd, outstanding_sd,'credit_limit_sd, bill_sd, remaining_bill_sd, payment_ratio_sd, overlimit_percentage_sd, payment_ratio_3month_sd, payment_ratio_6month_sd, delinquency_score_sd, years_since_card_issuing_sd, remaining_bill_per_number_of_cards_sd, remaining_bill_per_limit_sd,

total_usage_per_limit_sd, total_3mo_usage_per_limit_sd, total_6mo_usage_per_limit_sd, utilization_3month_sd, utilization_6month_sd, branch_code_A, branch_code_B, branch_code_C, branch_code_D, branch_code_E, branch_code_F, branch_code_G, branch_code_H, branch_code_I, branch_code_J, branch_code_K.

Implementation

After building a baseline model, I then attempted to train an XGBoost model using all available features, then planned to compare the performance to the baseline and another XGBoost model using certain features based on their correlation coefficient (see **Table 2**), that will rule out years_since_card_issuing, payment_ratio, payment_ratio_3month, payment_ratio_6month. The model's performance is evaluated by its ROC-AUC score on the testing data. As we had 3 models, we faced a challenge which was our models could be overfitting. Therefore we created a new model whose hyperparameter was adjusted so that overfitting could be reduced.

Refinement

The refinement process started with hyperparameter tuning. As XGBClassifier is compatible with scikit-learn framework, I was able to make use of scikit-learn's RandomSearchCV to perform tuning. The hyperparameter that would be optimised as follows:

- Min_child_weight: Minimum sum of weights of all observations required in a child node. Values = [1, 5, 10]
- Gamma: Minimum loss reduction for each node split. Values = [0.5, 1, 1.5, 2, 5]
- Subsample: Subsample ratio of the training instances. Values = [0.6, 0.8, 1.0]
- Colsample_bytree: Control the number of features to be sampled by each tree. Values = [0.6, 0.8, 1.0]
- Max depth: Maximum depth of a tree. Values = [3, 4, 5]

The hyperparameter tuning would optimise AUC score.

As we also experienced overfitting (see **Result**), we built a new model using also hyperparameter tuning, yet some hyperparameters above did not include in the parameter list will be tuned and some values were adjusted so we could achieve a model with reduced overfitting. Because our 2 prior XGBoost models got best parameters as follows:

- Model #2 XGBoost (all features)
  best_params: {'subsample': 0.6, 'min_child_weight': 1, 'max_depth': 5, 'gamma': 1.5, 'colsample_bytree': 0.8}
- Model #3 XGBoost (using selected features)
  best_params: {'subsample': 0.8, 'min_child_weight': 1, 'max_depth': 4, 'gamma': 1, 'colsample_bytree': 1.0}

To avoid overfitting, we need to set higher gamma and lower colsample_bytree, we set our gamma 2 and colsample_tree 0.6. We also update the Min_child_weight values from
[1, 5, 10] to [10, 50, 100]. The gap between AUC on the training set and test data for our final model was reduced.

IV. RESULT

After building the models, we tested them out by comparing their performance on the test dataset. Initially, we built

1. A baseline model using Logistic Regression algorithm
2. XGBoost using all values
3. XGBoost with features have low correlation to label discarded

	model	train_auc	test_auc	train_gini	test_gini
0	baseline (LR)	0.780434	0.770432	0.560868	0.540864
1	XGB with all features	0.966525	0.873918	0.933051	0.747835
2	XGB with selected features	0.923871	0.849346	0.847742	0.698691

Figure 5 AUC of models

XGB with all features shows better performance on test dataset, yet considering the performance on the train dataset, both XGB models may experience overfitting. Hence, we needed to build a new model where the hyperparameter would be tuned to avoid overfitting. Model #4 XGBoost used different hyperparameters and ruled out features with low correlation to the label. The following is the characteristics of the final model

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.8, gamma=2, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.02, max_delta_step=0, max_depth=5,
              min_child_weight=10, missing=nan, monotone_constraints='()',
              n_estimators=600, n_jobs=1, nthread=1, num_parallel_tree=1,
              random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
              subsample=1, tree_method='exact', use_label_encoder=False,
              validate_parameters=1, verbosity=None)
```

	model	train_auc	test_auc	train_gini	test_gini
0	baseline (LR)	0.780434	0.770432	0.560868	0.540864
1	XGB with all features	0.966525	0.873918	0.933051	0.747835
2	XGB with selected features	0.923871	0.849346	0.847742	0.698691
3	XGB avoid overfitting	0.910536	0.843357	0.821072	0.686714

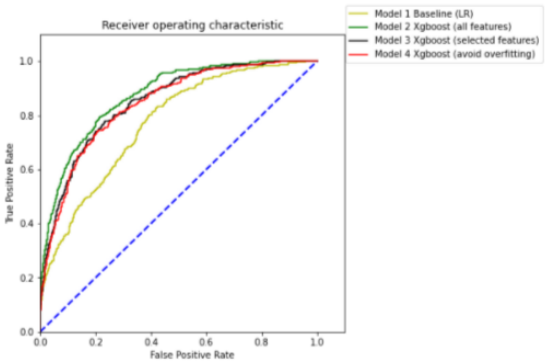


Figure 6 AUC of new model

Even though it shows that Model #2 has better performance on test dataset, yet Model #4 will be chosen as it has a reduced overfitting. The feature importance of Model #4 is shown below

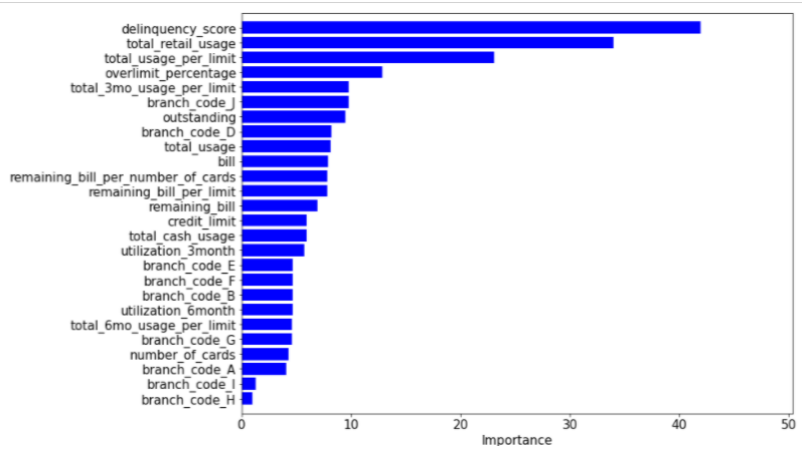


Figure 5 Feature Importance

This bar plot tells us the predictor relative contribution on each of the trees in Model #4 (all features used). delinquency_score, total_retail_usage and total_usage_per_limit are the highest contributors to the model.

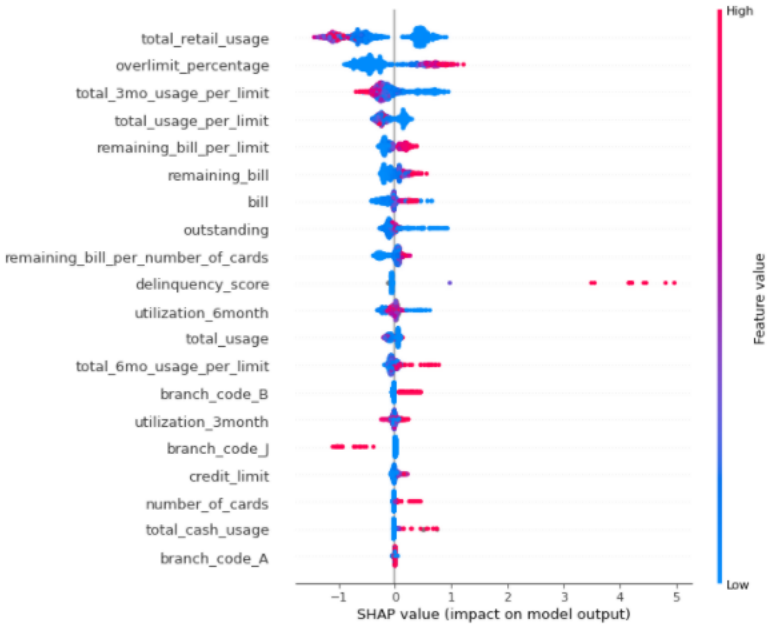


Figure 6 SHAP Values

:

Shap can help us explain the model easier, from the given picture, it shows that lower total_retail_usages, higher_overlimit_percentage, lower total_3mo_usage_per_limit can have higher default risk

V. CONCLUSION

- The final model has 0.84 AUC or 0.68 GINI. Below is the distribution of the default rate in bins. It shows that the higher score that machine learning predict a customer would have no ability to pay back, the more likely our prediction will be correct that the customer will actually not make payments.

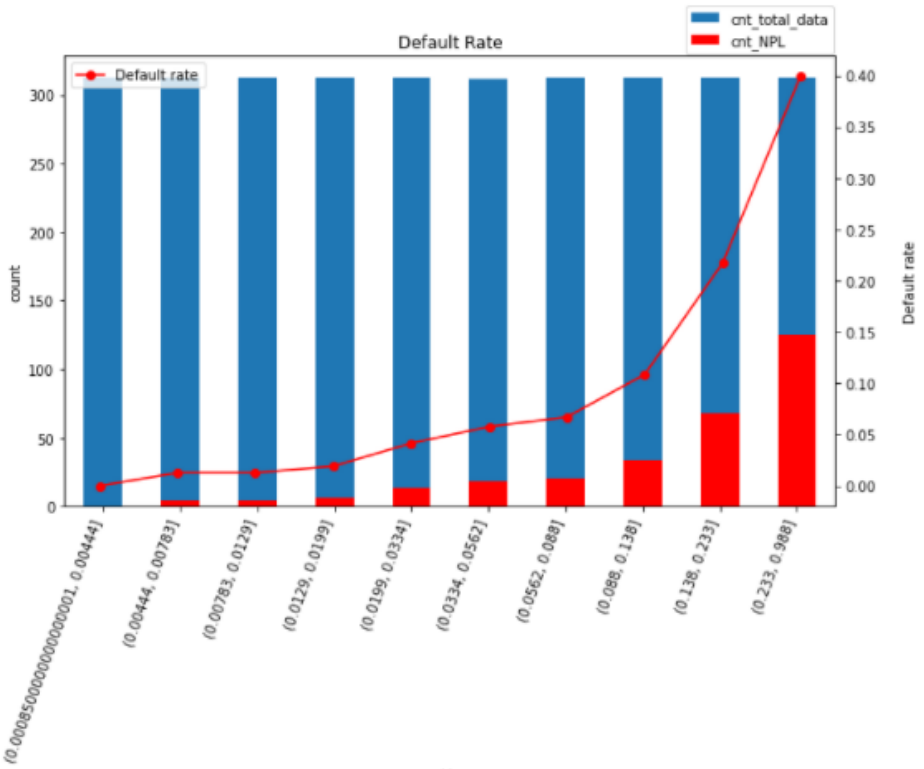
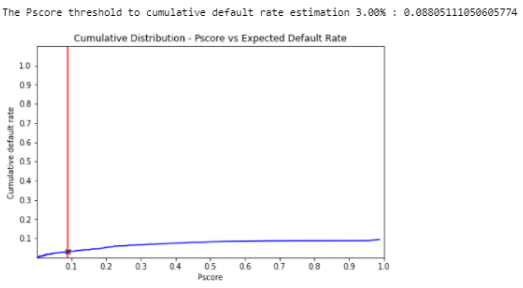


Figure 7 Bin plot

- The threshold for achieving default rate below 3% and approval rate higher than 70% = 0.08805



	score	approval_rate
0	0.08805	0.700224

Figure 8 Threshold search

- The credit scorecard that can help the nontechnical teams use our model aligning with their plan would be shown below. Example of using the scoring conversion table as follows, approve users with score less than 0.1 then we can get 73% approval rate and 3,2% default rate

	threshold	default_rate	approval_rate
1	0.1	0.032837	0.729946
2	0.2	0.052768	0.866091
3	0.3	0.065866	0.931608
4	0.4	0.074801	0.961329
5	0.5	0.082031	0.981783
6	0.6	0.085003	0.988814
7	0.7	0.086690	0.991691
8	0.8	0.087222	0.992969
9	0.9	0.087222	0.992969
10	1.0	0.093640	1.000000

Figure 9 Scorecard

Improvement

- More feature engineering could be performed to have a better model, such as trying any other encoding methods or any other missing value handlings.
- There are a lot of Machine Learning algorithms that we haven't tried and maybe they would perform better than our solution.

VI. REFERENCE

- https://towardsdatascience.com/how-to-develop-a-credit-risk-model-and-scorecard-91335fc01f03
- https://medium.com/henry-jia/how-to-score-your-credit-1c08dd73e2ed
- https://towardsdatascience.com/machine-learning-gridsearchcv-randomizedsearchcv-d36b89231b10