PACMANN Intro to Machine Learning Project: A Random Forest Model to Determine Credit Approval

Muhammad Alif Aqsha

July 9, 2023

Contents

1 Background

A credible credit risk scoring is crucial for financial institutions, especially those who lend money. This system allows them to quantify and estimate the risk that loan applicants carry; whether it is the risk of defaulting or serious late payments such that high-risk applicants may be flagged and given a higher interest rate to account for their risks or be rejected at all.

It is necessary for financial institutions to be able to balance their selection criteria; too restrictive, and they might reject a large number of sound applicants; too lax, and they might allow for a large number of NPLs (non-performing loans). One solution for financial institutions to accurately predict whether a lender will pay their loan in time or not is to develop a machine learning model. In contrast to traditional programming, the output of such an ML algorithm is a rule on how to approve or reject a credit application.

This ML model will be fitted on some training dataset consisting of loan applicants' financial and personal information (income, total credit balance divided by total credit limit, debt ratio, age, etc) and target variable whether they repaid the loan in time or not. This produces a fitted ML model that will hopefully perform well in test datasets (able to separate good loan applicants from bad ones). For this project, we will use a Random Forest model.

2 Dataset & Features

In this project, we are going to use "cs-training.csv" and split it into training and test sets. For validation, we are going to use "cs-test.csv" (which doesn't have the target variable) and submit it into Kaggle.

The data features a target column 'SeriousDlqin2yrs' describing whether the applicant had experienced 90 days (or worse) late payments at the present.

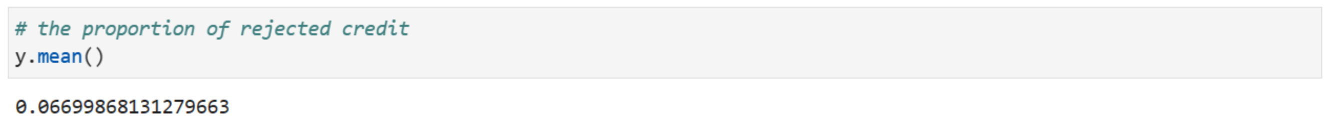
The data consists of the financial conditions (debt ratio, total credit balance divided by total credit limit, income, etc) and their personal information (age and number of dependents).

Variable Name	Description	Type
RevolvingUtilizationOfUnsecuredLines	Total balance on credit cards and personal lines of credit (excluding real estate and installment debt like car loans) divided by the sum of credit limits	float
Age	Age of borrower in years	integer
NumberOfTime30-59DaysPastDueNotWorse	Number of times applicants has experienced 30-59 days late payments in the last 2 years (before the 'SeriousDlqin2yrs' period)	integer
DebtRatio	Monthly debt payments, alimony, living costs divided by monthy gross income	float
MonthlyIncome	Monthly income	float
NumberOfOpenCreditLinesAndLoans	Number of open loans (installments like car loans or mortgages) and lines of credit (e.g. credit cards)	integer
NumberOfTimes90DaysLate	Number of times applicants have experienced 90 days or more late payments in the last 2 years (before the 'SeriousDlqin2yrs' period)	integer
NumberRealEstateLoansOrLines	Number of mortgages and real estate loans (including home equity) lines of credit	integer
NumberOfTime60-89DaysPastDueNotWorse	Number of times applicants have experienced 60-89 days late payments in the last 2 years (before the 'SeriousDlqin2yrs' period)	integer
NumberOfDependents	Number of dependents in the family excluding themselves (spouse, children etc.)	integer

3 Data Preprocessing

First, we import the data and drop any duplicates. Then, before we impute any missing or problematic values, we first separate the original data into training and test set.

The proportion of rejected application is about 0.06, so this is a case of imbalanced class.

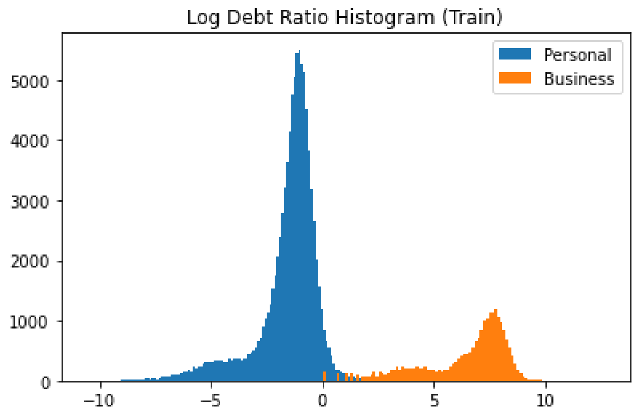


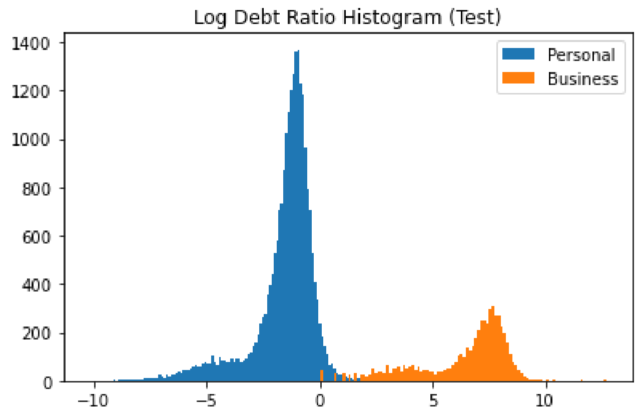
```
# the proportion of rejected credit
y.mean()
```

```
0.06699868131279663
```

The variables which include null values are only MonthlyIncome and NumberOfDependents. For MonthlyIncome, it is very likely that null values represent a different type of loan applicants, most likely business ones. The reason for this is that for these applicants, their DebtRatio's are exact integers (unlike those with non-null MonthlyIncome).

Furthermore, their DebtRatio distribution is very different too.





As such, we might introduce a new TypeCustomer column derived from MonthlyIncome; the value is 0 if it has a non-null MonthlyIncome, and 1 otherwise. Then we impute missing MonthlyIncome (the Business customers) with 0.

Now, we should identify unusual (potentially mistaken) values from the columns. We expect DebtRatio to range between 0 and 1, but there exists many large values in the data. But this might be explained by two reasons: It is normal for businesses to apply for an extremely large sum of loans.

For personal loans with big DebtRatio, it is likely that they came from student loans (students or recent-graduates are likely to have 0 or very little income). Then, we also expect RevolvingUtilizationOfUnsecuredLines to range between 0 and 1, but there also exists a substantial amount of large values in the data. But this might be explained as follows:

1. For values greater than 1 (but not extremely large), it is still possible for borrowers to exceed their credit limits.
2. For extremely large ones, it is possible that the borrowers might have closed their credit cards but still have an outstanding invoice. Thus, their credit balance values are large while their credit limits are 0 (perhaps to avoid undefined division, the system computes RevolvingUtilizationOfUnsecuredLines by dividing credit balance with (1 + credit limits).

Another insight why the system might compute RevolvingUtilizationOfUnsecuredLines by dividing credit balance with (1 + credit limits): after checking with a simple code (not included in the Jupyter Notebook), all DebtRatio with non-null MonthlyIncome values are a multiple of 1/(1+MonthlyIncome).

Now, for the age column, there exists a 0-year-old applicant with NumberOfDependents value of 2, which does not make any sense. We impute this value with the median age (from the training set).

For NumberOfTime30-59DaysPastDueNotWorse column, we found unusually large values (96 & 98 to be exact). There exists a 21-year-old having this column equals 98. It doesn't make any sense for a 21-year-old to have 98 times of 30-59 late payments ($98 \cdot 30/365 = 8.05$ years); it doesn't make sense for a 21-year-old to have started making credit payment from the age 13. Due to this, we impute these large values with the mode from the training set (which is 0). The same applies for the column NumberOfTime60-89DaysPastDueNotWorse and NumberOfTimes90DaysLate columns.

4 Method

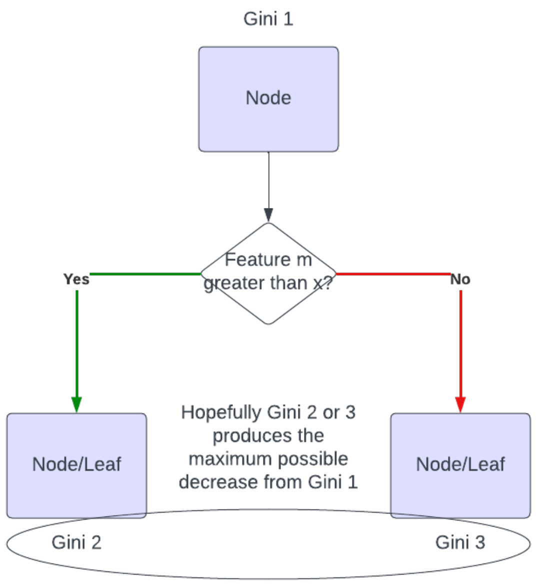
As we have mentioned in the Background section, we are going to use a random forest model consisting of numerous smaller decision trees.

A decision tree consists of several "nodes". At each node, the algorithm branches the input training data into two groups according to exactly one feature threshold. The branching rule (i.e. the feature and threshold used) is determined such that it maximises the decrease in Gini Impurity measure

$$Gini = 1 - \sum_{i \in \text{class}} p_i^2 \tag{1}$$

where p_i is the proportion of class-i observations. If the observations belong mostly only to one class, then $\sum\limits_{i \in \text{class}} p_i^2$ is close to 1 (so the Gini measure is close to 0); otherwise, if they are more random, then the formula is close to $\sum\limits_{i \in \text{class}} \frac{1}{|\text{class}|^2} = \frac{1}{|\text{class}|}$ (here $|\text{class}|$ denotes the number of classes), producing a Gini measure close to $1 - \frac{1}{|\text{class}|}$.

Aside from nodes, a decision tree also consists of several "leaves" where there is no more branching. At these leaves, we expect that the Gini measure is as minimum as possible (close to 1) as it indicates that the observations group is "pure" i.e. almost all of them belonged to a certain class. Then, all observations (present or future) will be classed as this certain class if they fall into this leaf.


Gini 1
Node
Feature m greater than x?
Yes
No
Node/Leaf
Hopefully Gini 2 or 3 produces the maximum possible decrease from Gini 1
Node/Leaf
Gini 2
Gini 3

A random forest classifier build many small decision trees (with shallow depth) trained only on a subset of bootstrapped data and randomly-selected features. It then computes the probability that an observation falls under a class by calculating the portion of decision trees where the observation falls under that class.

Model Training

```
For $j = 1, ..., num\_trees$ do:
    1. Generate $X$_bootstrap, $y$_bootstrap by bootstrapping a portion of training observations
    2. If necessary, undersample $X$_bootstrap, $y$_bootstrap, producing $X$_res, $y$_res
    3. Randomly choose $m$ features
    4. Fit a decision tree decision_tree$_j$ using $X$_bootstrap, $y$_bootstrap (or $X$_res, $y$_res) and the selected features.
    5. Save decision_tree$_j$
Predicting Probability
    1. Input feature values
    2. For $j = 1, ..., num\_trees$ do:
        (a) Predict using decision_tree$_j$
    3. For each class i, do:
        (a) Calculate the proportion of predictions falling under the $i$ class
```

5 Discussion

5.1 Hyperparameter

In this project, we only concern on the individual tree depths. We chose the maximum depth to be 5 as to avoid overfitting (large variance) and avoid long computations. As such, we do not perform a cross-validation to determine the best maximum depth possible.

5.2 Relevant Metrics

In this project, we are going to use the AOC (area-under-curve) of ROC (receiver operating characteristics) curve, as we might encounter an imbalance target column (normally, there are only a few rejected loan applicants). ROC plots the false positive rate versus the true positive rate.

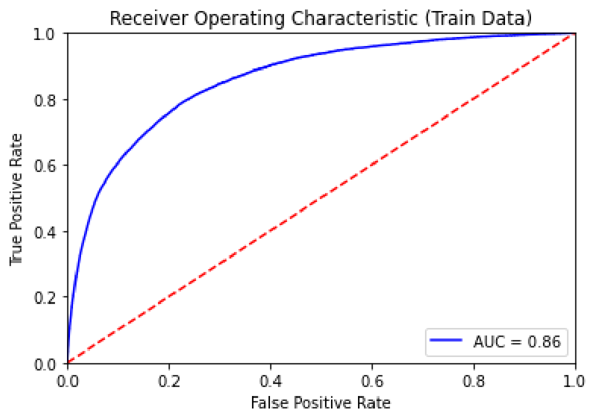
$$\begin{aligned} TPR &= \frac{\text{\# rejected actual bad loans}}{\text{\# actual bad loans}} \\ FPR &= \frac{\text{\# rejected actual good loans}}{\text{\# actual good loans}} \end{aligned} \tag{2}$$

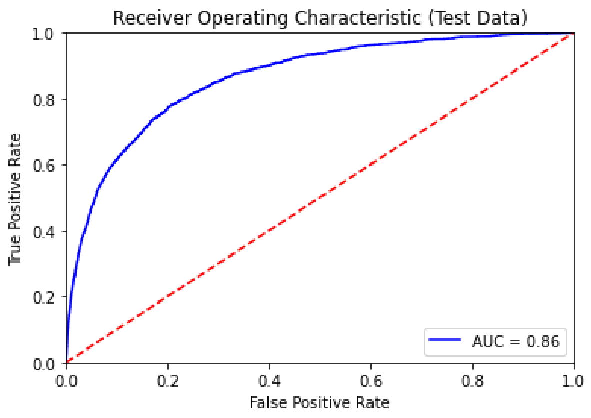
An AUC of 0.5 or less indicates a bad model as it cannot truly separate the classes, while a value of 0.7-0.8 is generally considered acceptable, 0.8-0.9 is excellent, and above 0.9 is considered outstanding (Hosmer & Lemeshow, 2000).

Why do we use this metric? Obviously, financial lenders will try to accept as many good loan applications as possible to maximise their revenue from interests, while rejecting almost all bad ones to avoid losses from NPLs (non-performing loans). Because of this, we want to maximise true positive rate (reject a high portion of actual bad loans) while maintaining a low false positive rate (reject a low portion of actual good loans).

5.3 Results

Here are the ROC curves for the training and test dataset.





As we might see, we obtain a very good AUC score of 0.86 for both training and test data.

Here is the classification report on the test data if we set threshold = 0.5. The table for the training data is more-or-less similar.

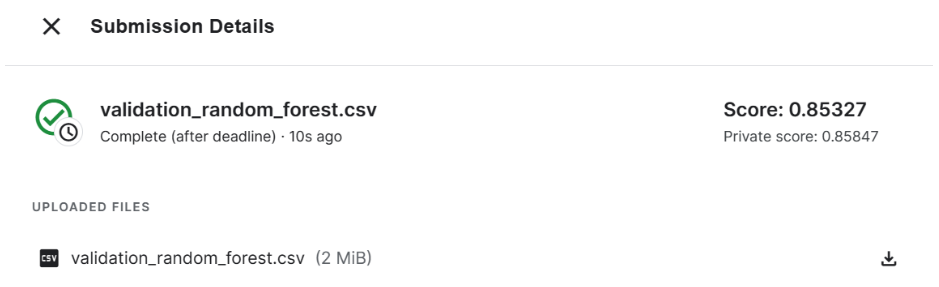
```
              precision    recall  f1-score   support

    Approved       0.98      0.80      0.88     27877
    Rejected       0.22      0.77      0.34      2002

    accuracy                           0.80     29879
   macro avg       0.60      0.78      0.61     29879
weighted avg       0.93      0.80      0.84     29879
```

The precision of predicting class ”Rejected” is 0.22, meaning that only 78% of rejected loans are actually good loans. But this does not really matter as the model only reject about 23% of loans. Moreover, the recall of predicting class ”Approved” is 0.80, meaning that only 20% of actual good loans are rejected. Furthermore, the recall of class ”Rejected” is 0.77, meaning that the model can reject 77% of actual bad loans.

If we validate this model and submit the validation prediction to Kaggle, we obtain a similar AUC score. To give some context, the highest-ranked participant obtained an AUC score of 0.86955.



Submission Details

validation_random_forest.csv
Complete (after deadline) · 10s ago

Score: 0.85327
Private score: 0.85847

UPLOADED FILES

validation_random_forest.csv (2 MiB)

6 Future Works

1. The training might be separated into 2 models: model with valid monthly income values (TypeCustomer = 0) and model with undefined monthly income (TypeCustomer = 1).
2. We might try an adaptive-boosting approach where the class weights are updated on each sub-models.
3. We might simulate the effect of the number of sub-models (estimators), proportion of bootstrapped observations and number of selected features on the AUC score and see at which point the model stabilise or reach the maximum performance.

7 References

Hosmer, D. W., S. Lemeshow (2000). *Applied Logistic Regression*. 2nd ed. John Wiley and Sons.

Data Source: Give Me Some Credit — Kaggle

Github Repository: PACMANN Intro to ML Final Project