Team 12
Model Stacking for Credit Card Approval Classification

Abhishek Balaji, Au Jun Hui, Fong Ken Rui, Sang Yong En Sean

Introduction

Credit cards are issued by financial institutions (FI) that allow cardholders to make purchases on credit. In return, FIs earn money via interest on overdue payments. As such, there is a vested interest for FIs to ensure that the cardholder has the ability to repay the amount borrowed on credit. Our project utilizes different Artificial Intelligence (AI) models, such as Logistic Regression (LR), Decision Trees (DT), Random Forest Regression (RF), and Light Gradient-Boosting Machine (LightGBM), and combine them through model stacking to do binary classification which indicates whether a particular application should be approved or rejected.

Related Works

Using Logistic Regression, Linear Support Vector classification and Naive Bayes Classifier, Zhao (2022) was able to achieve ˜90% balanced accuracy and ˜89% accuracy rate using the Linear SVC model in predicting credit card approval, albeit using a different dataset. Another study by Peela *et al.* (2022) used Logistic Regression and Random Forest Regression, achieving an accuracy of 86% on a separate dataset.

Outside of academic papers, there are also notebooks present on Kaggle and Github using the same dataset, but their implementations are mostly flawed with the presence of information leakage and incorrect pre-processing techniques.

Dataset

The datasets used consist of a credit card 'applications' dataset and a monthly 'credit record' dataset. Each unique ID in both datasets refers to a unique person. There are a total of 438,557 and 1,048,575 entries in the 'applications' and 'credit record' dataset respectively. Each application contains personal information of the applicant, which is a mixture of both categorical and continuous data. On the other hand, each unique ID in the credit record is associated with a series of monthly records. Each record contains a categorical label that describes the monthly credit status of the credit cardholder.

For the 'applications' dataset, 0.01% of applications are non-unique. This small proportion will probably have an insignificant impact on the model, hence we remove these rows. The data appears to be relatively clean, with the exception of the 'job' column where more than 25% of the entries are missing (Appendix B Fig.3), and is thus dropped.

The correlation matrix (Appendix B Fig. 4) indicates that the num_child feature is highly correlated with CNT_FAM_MEMBERS and thus, we drop the less generic num_child feature. As the remaining features are relatively uncorrelated, our models can be applied with assumed feature independence.

8.3% of unique applications share common IDs with the 'credit record' dataset, yielding a combined dataset with 36,457 unique IDs. An examination of the distribution of records show that users with good credit statuses ('X', 'C', '0') are heavily over-represented in the data (Appendix A Fig. 5). Therefore, it is likely that an imbalance in the target variable will be introduced in our feature engineering.

Feature Engineering

Credit Records

With the objective of using the credit records to generate the target variable, we simplify the records into a risk score for each ID. By giving a score of 1 to statuses of 'X' ,'C' and '0', 0 to '1' and '2', and -1 to '3', '4', and '5', a final risk score is generated with the mean of the scores for each ID. A binary classification is applied to the results, with a threshold mean of more than 0.905 being 1 (representing 'Good' applicants), and any value lower than that being a 0 (representing 'Bad' applicants).

Applications

Categorical data in the 'applications' dataset is handled by splitting the categories into dummy variables. The 'num_child' and 'jobs' variables are dropped based on the aforementioned data analysis. The 2 datasets are then merged using an inner-join, resulting in a dataset where only applications with credit histories are present. This allows us to utilize our target variable for training and testing (36,457 rows). Based on our risk score and threshold, 5.32% of the applicants were found to be 'Bad' applicants, mirroring current delinquency rates according to CNBC (2022). As the

resulting dataset is very imbalanced, we must take this into account when training our models.

Methods

We formulate our problem as aiming to minimize the rejection of 'Good' applicants, or False Negative Rate (FNR), while maximizing the rejection of 'Bad' applicants, or True Negative Rate (TNR). To achieve this, the rationale for choosing our models are as follows:

- LR is easy to implement, efficient to train and because of gradient descent, the model can also be updated easily if certain features are dropped from the dataset.
- DTs tend to have high accuracy and low bias as the model makes only few assumptions about the target function. However, the low bias comes at the cost of high variance because the decision tree is highly sensitive to changes in the node parameters and changes to the dataset.
- RF regression overcomes the shortfalls of DTs through feature bagging which decorrelates the features chosen in each split as noted in Kho (2018), resulting in a lower variance in exchange for a slightly higher bias than DTs.
- LightGBM is a gradient-boosting framework that is efficient and accurate because of its use of Gradient-Based One-Side Sampling. The model attains its high accuracy and efficiency by dropping data with small gradients, leaving a smaller amount of data with high information gain to work with according to Ke *et al.* (2017).

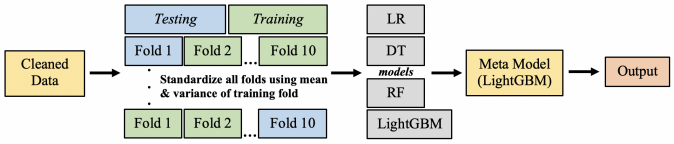


Figure 1: Flowchart of our implementation

Training Methodology

We used stratified 10-fold cross validation to split the training data to evaluate our model generally by reducing overfitting. It has been shown in James *et al.* (2017) that the bias and variance are reduced when using 10 folds. To prevent information leakage, we standardize our training dataset in each iteration, and use the parameters of the training set to standardize the testing set.

When evaluating our models, we plot the confusion matrix, receiver operating characteristic curve (ROC curve), precision-recall curve and classification report.

We first create our baseline models by implementing them on the cleaned dataset without any remedial measures to correct the imbalanced data (Appendix C). Subsequently, to account for the imbalanced data, our first remedial measure is to apply Synthetic Minority Over-sampling Technique to over-sample the minority class such that the proportion of both classes becomes equal in the training dataset (Appendix D). For the second remedial measure, we utilize class weights to adjust the cost function of the model so that it is biased against the misclassification of the minority class during training. (Appendix E)

Model Stacking

Since the results of the baseline models and the 2 remedial measures show that the models have limited effectiveness in maximizing TNR, we implemented model stacking with balanced class weights to improve our model by combining the outputs of multiple individual models and running them through another model called a 'meta-learner' (Fig. 1).

According to Pedersen (2021), the meta-learner maximizes and minimizes the strengths and weaknesses of each model. Consequently, the resulting model usually generalizes well on unseen data.

Results and Discussions

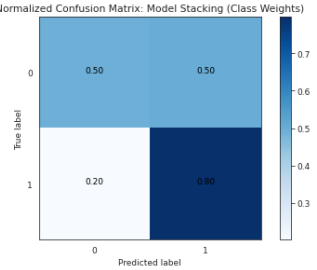


Figure 2: Confusion matrix of the stacked model

Looking at the confusion matrix (Fig. 2), other evaluation metrics in the classification report, the ROC curve and the precision-recall curve (Appendix A), we can see that our model performs significantly better than the previously implemented individual models.

For example, the confusion matrix of the baseline LightGBM model (Appendix C) indicates that the model was classifying every applicant as 'Good', with a TNR and FNR of 0. This means that the model cannot recognise what constitutes a 'Bad' application.

With model stacking, we now have a TNR of 50% and a FNR of 20%. We believe that this model best achieves our problem formulation mentioned earlier. As an aside, we adjusted the hyper-parameters of each model within the model stacking process, but did not see any significant improvement to the results. Hence, the only hyper-parameter used was having balanced class weights in all the models.

Conclusion

Our model was developed based on a number of assumptions. Firstly, we developed our own metric in order to come up with the target variable, since the data was unlabelled. FIs wishing to implement our model to help with their credit card approval process should use their own approval criteria in order to categorise a 'Good' and 'Bad' applicant. Our problem formulation may also differ in a practical setting based on the amount of profit earned by the FI.

Overall, we believe that our project has achieved its goals set out in our problem formulation, and that it serves as a foundation for developing more complex models to tackle this issue of Credit Card Approval.

References

James G, Witten D, Hastie T, Tibshirani R (2017) 5.1.4 Bias-Variance Trade-Off for k-Fold Cross-Validation. In: An introduction to statistical learning: With applications in R. Springer, New York, NY, pp 205–206

Ke, Guolin; Meng, Qi; Finley, Thomas; Wang, Taifeng; Chen, Wei; Ma, Weidong; Ye, Qiwei; Liu, Tie-Yan (2017). "LightGBM: A Highly Efficient Gradient Boosting Deci-sion Tree". Advances in Neural Information Processing Systems. 30.

Kho J (2018) Why Random Forest is My Favorite Machine Learning Model. In: Towards Data Science. https://towardsdatascience.com/why-random-forest-is-my-favorite-machine-learning-model-b97651fa3706. 2022

Pedersen T (2022) How to use "Model stacking" to improve machine learning predictions. In: Medium. https://medium.com/geekculture/how-to-use-model-stacking-to-improve-machine-learning-predictions-d113278612d4. 2022

Peela HV, Gupta T, Rathod N, et al (2022) Prediction of Credit Card Approval. International Journal of Soft Computing and Engineering (IJSCE)

Singh K (2022) How to dealing with imbalanced classes in machine learning. In: Analytics Vidhya. https://www.analyticsvidhya.com/blog/2020/10/improve-class-imbalance-class-weights/. 2022

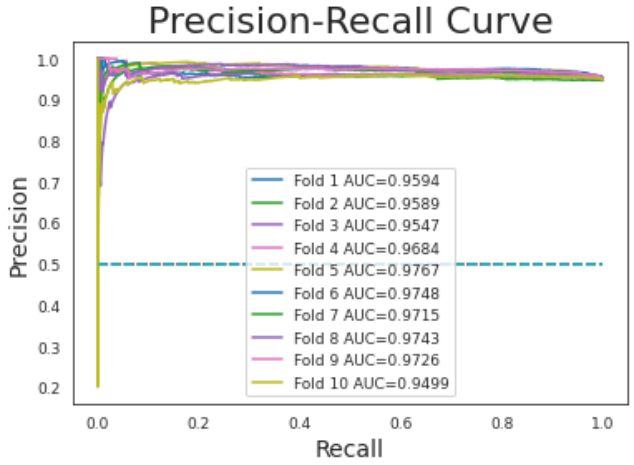
White A (2022) Credit card debt in the U.S. hits all-time high of $930 billion-here's how to tackle yours with a balance transfer. In: CNBC. https://www.cnbc.com/select/us-credit-card-debt-hits-all-time-high/. 2022

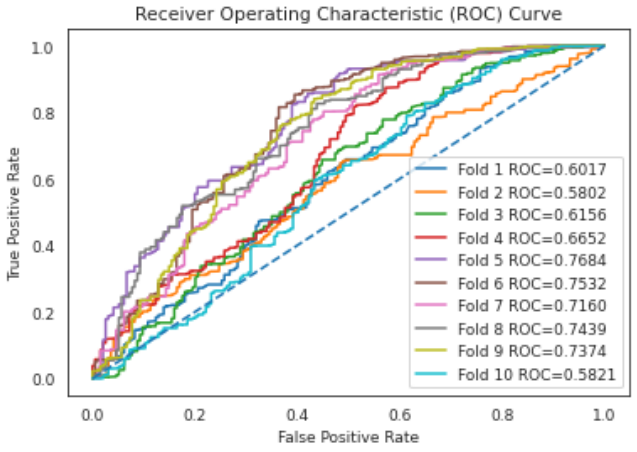
Zhao Y (2022) Credit card approval predictions using logistic regression, linear SVM and naïve Bayes classifier. 2022 International Conference on Machine Learning and Knowledge Engineering (MLKE). doi: 10.1109/mlke55170.2022.00047

Appendix A: Additional results of Optimised Model (Model Stacking)

Note: Results below may differ slightly from the results seen when you run the source code. This is due to the random sampling in stratified k-fold cross validation.

	0	1	accuracy	macro avg	weighted avg
precision	0.118715	0.965061	0.776147	0.541888	0.919957
recall	0.490352	0.792231	0.776147	0.641291	0.776147
f1-score	0.190693	0.869871	0.776147	0.530282	0.833676
support	194.300000	3451.400000	0.776147	3645.700000	3645.700000





Appendix B: Charts from Exploratory Data Analysis

Figure 5: Histogram plots of the dataset features

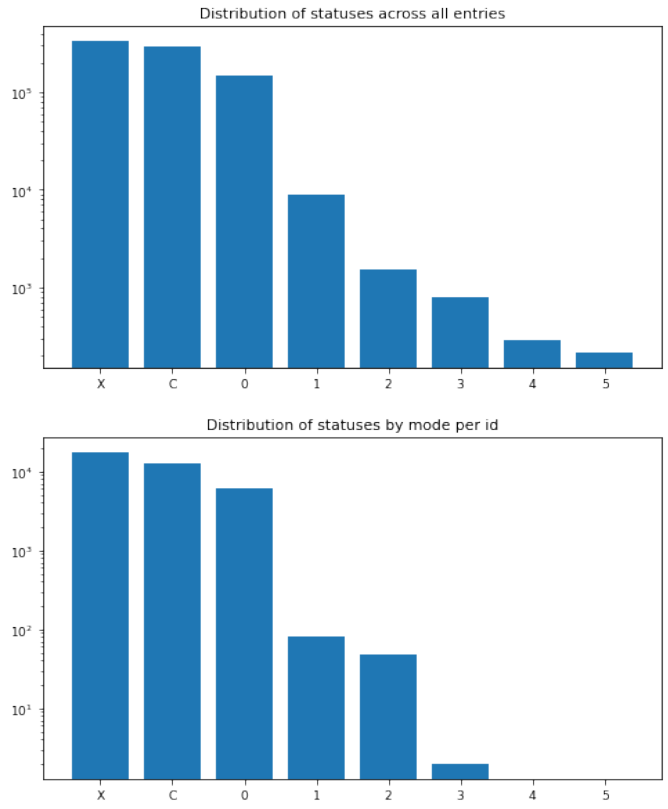


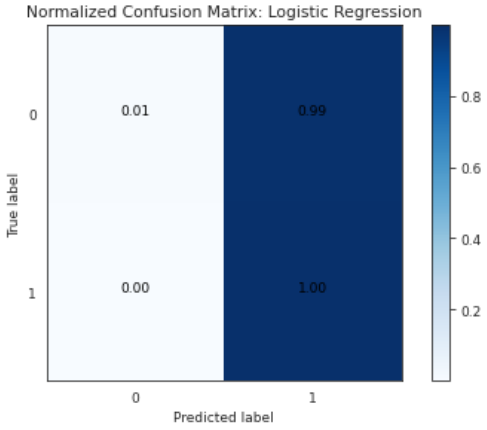
Figure 6: Histogram of total credit statuses as well as mode credit statuses aggregated by ID

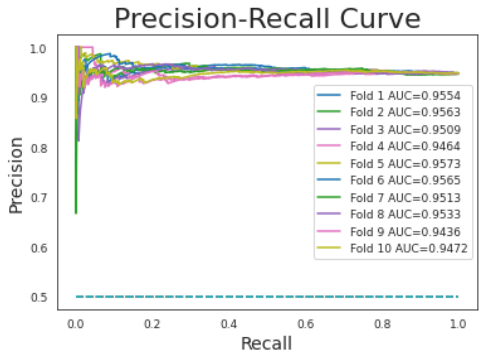
Appendix C: Results of Baseline Models

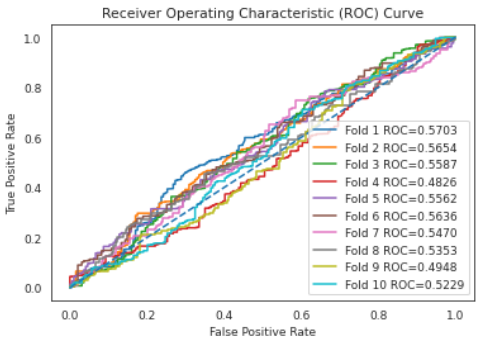
Note: Results below may differ slightly from the results seen when you run the source code. This is due to the random sampling in stratified k-fold cross validation.

Logistic Regression (LR)

	0	1	accuracy	macro avg	weighted avg
precision	0.158333	0.946957	0.946842	0.552645	0.904916
recall	0.005155	0.999855	0.946842	0.502505	0.946842
f1-score	0.009842	0.972687	0.946842	0.491265	0.921371
support	194.300000	3451.400000	0.946842	3645.700000	3645.700000

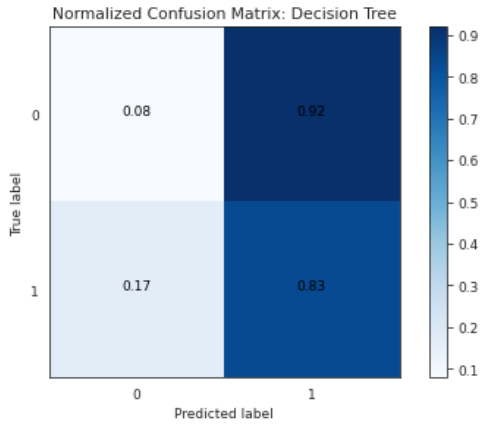


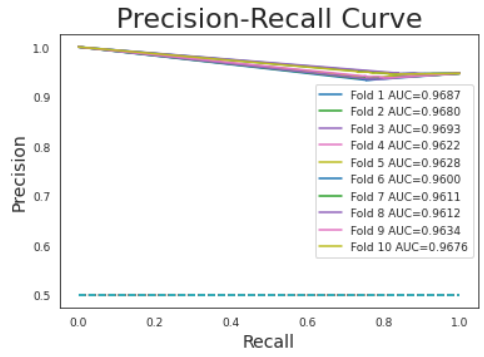


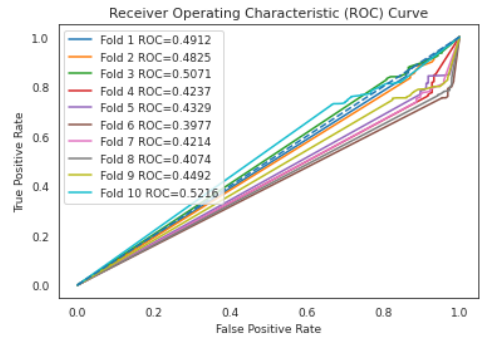


Decision Tree (DT)

	0	1	accuracy	macro avg	weighted avg
precision	0.029082	0.941373	0.792356	0.485227	0.892751
recall	0.081353	0.832381	0.792356	0.456867	0.792356
f1-score	0.041807	0.882942	0.792356	0.462375	0.838114
support	194.300000	3451.400000	0.792356	3645.700000	3645.700000

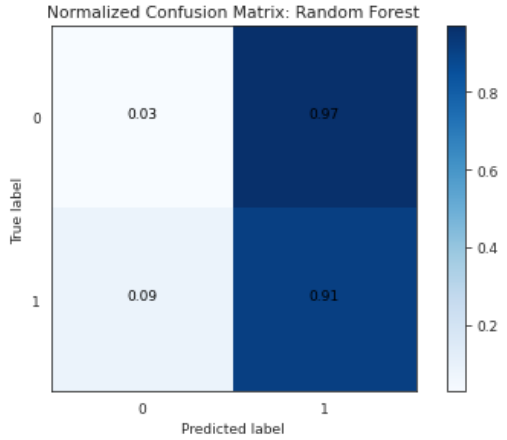


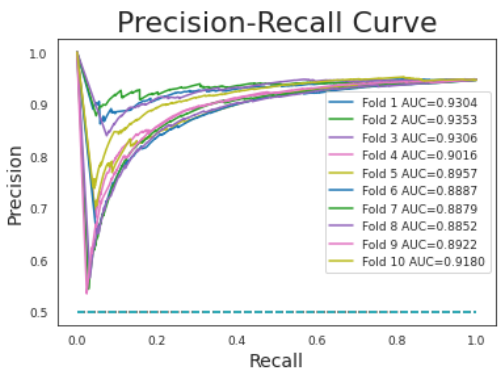


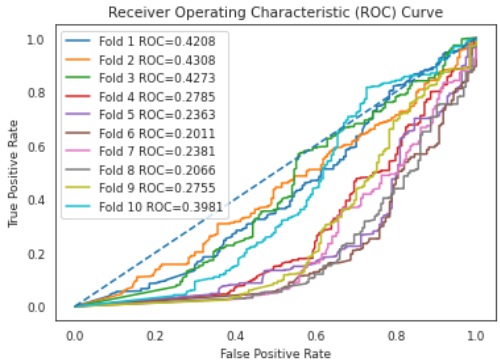


Random Forest (RF)

	0	1	accuracy	macro avg	weighted avg
precision	0.025696	0.943396	0.864221	0.484546	0.894486
recall	0.030381	0.911162	0.864221	0.470771	0.864221
f1-score	0.024368	0.926579	0.864221	0.475474	0.878496
support	194.300000	3451.400000	0.864221	3645.700000	3645.700000

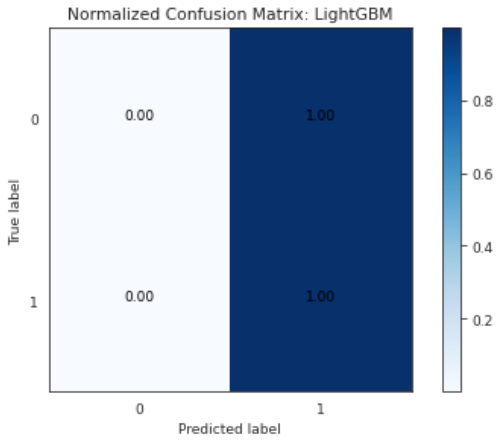


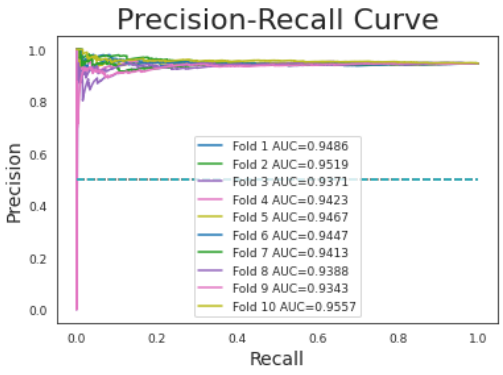


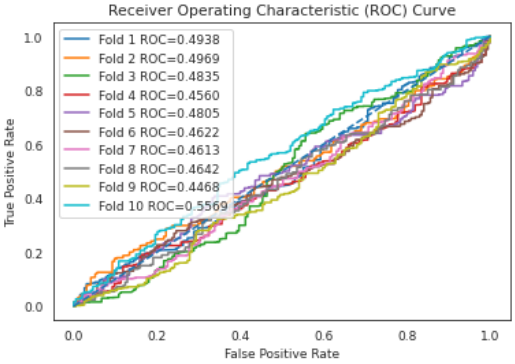


Light Gradient Boosting Machine (LightGBM)

	0	1	accuracy	macro avg	weighted avg
precision	0.038571	0.946594	0.943275	0.492583	0.898197
recall	0.001546	0.996291	0.943275	0.498919	0.943275
f1-score	0.002970	0.970805	0.943275	0.486888	0.919223
support	194.300000	3451.400000	0.943275	3645.700000	3645.700000





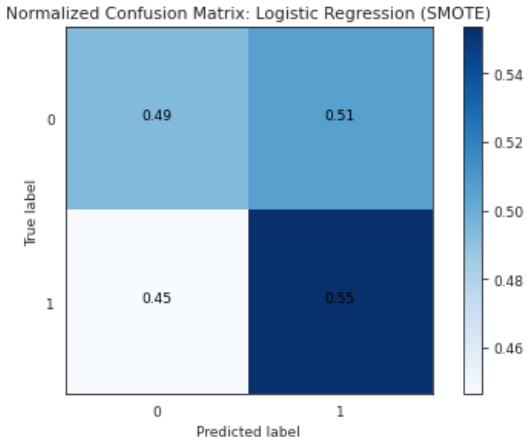


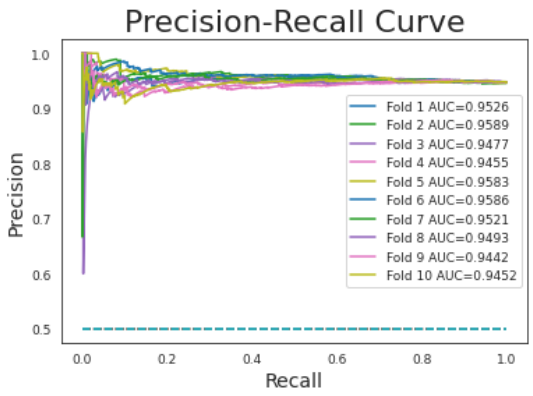
Appendix D: Results of Remedial Measure 1: SMOTE

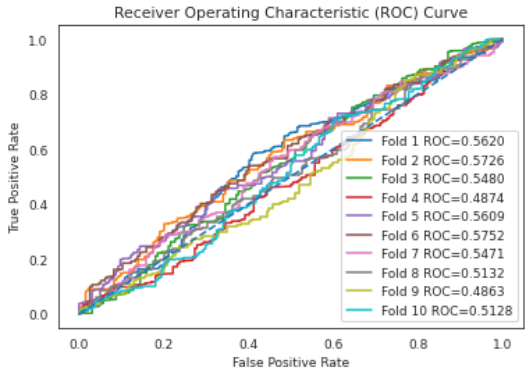
Note: Results below may differ slightly from the results seen when you run the source code. This is due to the random sampling in stratified k-fold cross validation.

Logistic Regression (LR)

	0	1	accuracy	macro avg	weighted avg
precision	0.059327	0.950632	0.550205	0.504980	0.903130
recall	0.494073	0.553365	0.550205	0.523719	0.550205
f1-score	0.105813	0.698122	0.550205	0.401967	0.666554
support	194.300000	3451.400000	0.550205	3645.700000	3645.700000

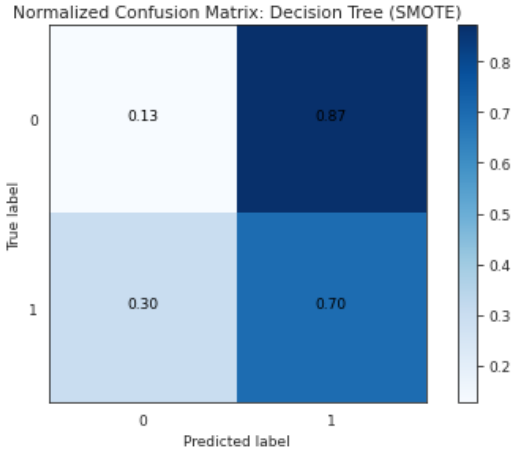


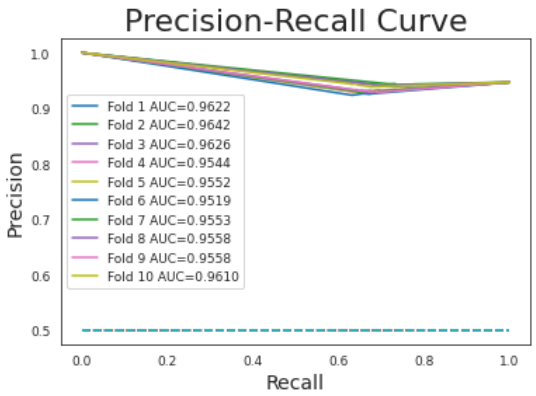


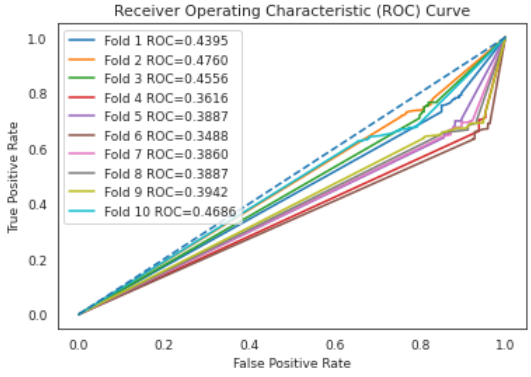


Decision Tree (DT)

	0	1	accuracy	macro avg	weighted avg
precision	0.024039	0.934275	0.668786	0.479157	0.885763
recall	0.128723	0.699190	0.668786	0.413956	0.668786
f1-score	0.040399	0.799318	0.668786	0.419858	0.758871
support	194.300000	3451.400000	0.668786	3645.700000	3645.700000

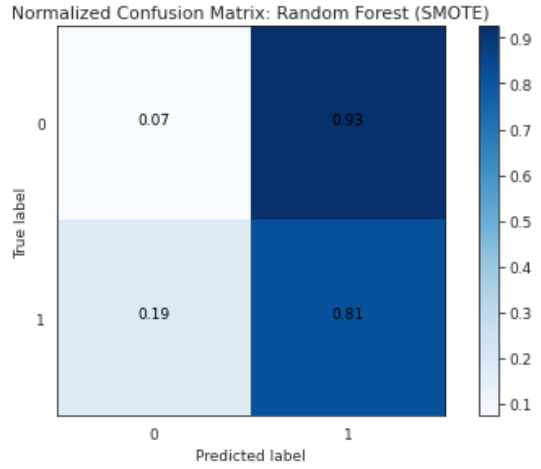


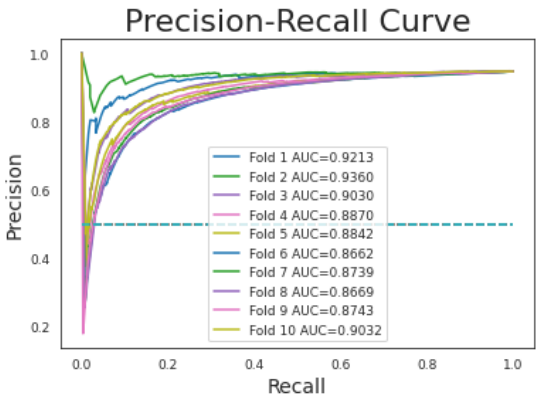


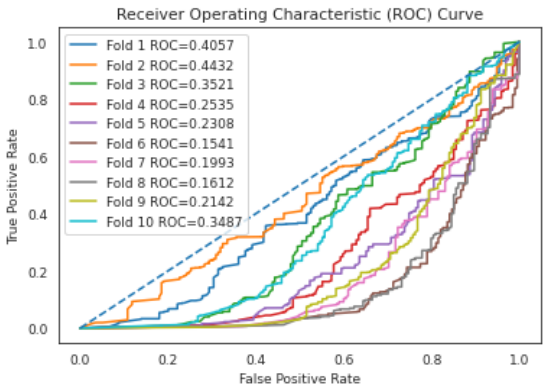


Random Forest (RF)

	0	1	accuracy	macro avg	weighted avg
precision	0.022738	0.939693	0.774471	0.481215	0.890822
recall	0.074155	0.813896	0.774471	0.444026	0.774471
f1-score	0.034217	0.871653	0.774471	0.452935	0.827022
support	194.300000	3451.400000	0.774471	3645.700000	3645.700000

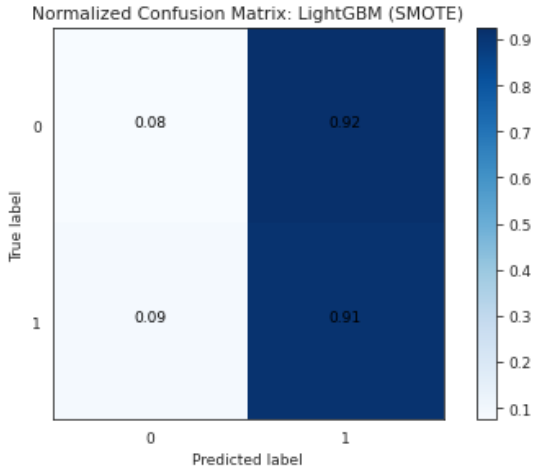


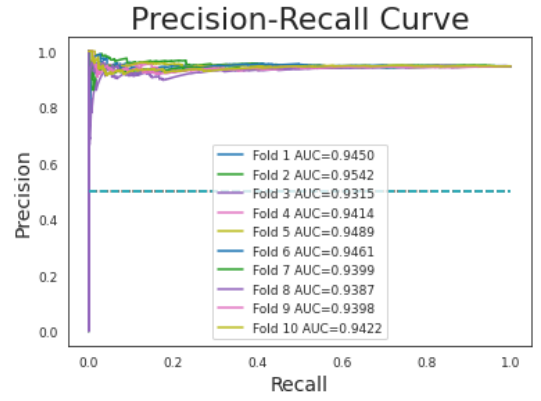


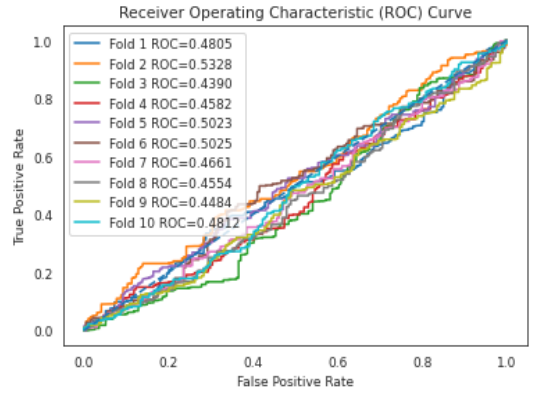


Light Gradient Boosting Machine (LightGBM)

	0	1	accuracy	macro avg	weighted avg
precision	0.046367	0.945994	0.866609	0.496181	0.898048
recall	0.076183	0.911108	0.866609	0.493645	0.866609
f1-score	0.057435	0.928181	0.866609	0.492808	0.881774
support	194.300000	3451.400000	0.866609	3645.700000	3645.700000





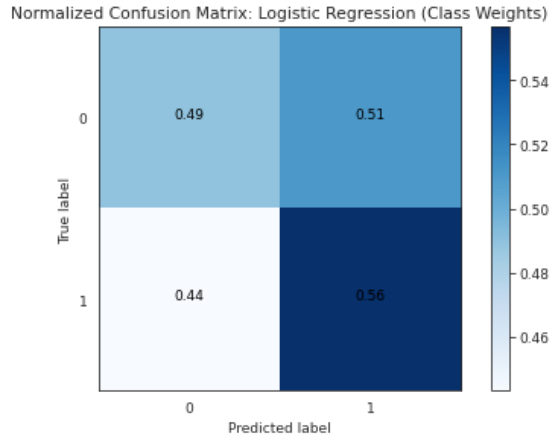


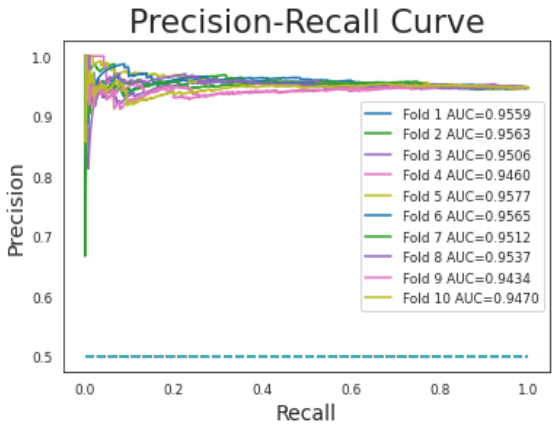
Appendix E: Results of Remedial Measure 2: Class Weights

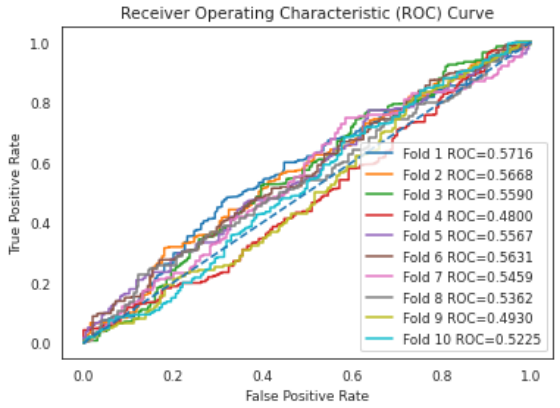
Note: Results below may differ slightly from the results seen when you run the source code. This is due to the random sampling in stratified k-fold cross validation.

Logistic Regression (LR)

	0	1	accuracy	macro avg	weighted avg
precision	0.059084	0.950522	0.553139	0.504803	0.903013
recall	0.488937	0.556755	0.553139	0.522846	0.553139
f1-score	0.105273	0.700440	0.553139	0.402856	0.668719
support	194.300000	3451.400000	0.553139	3645.700000	3645.700000

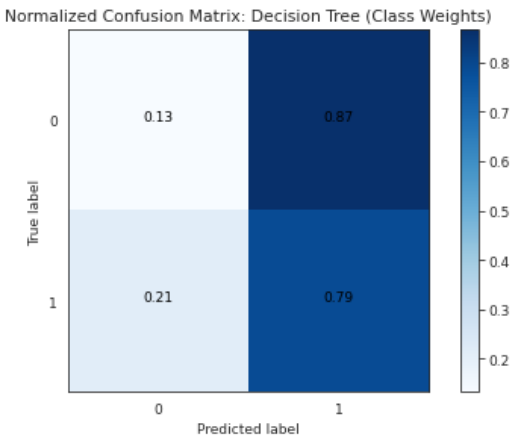


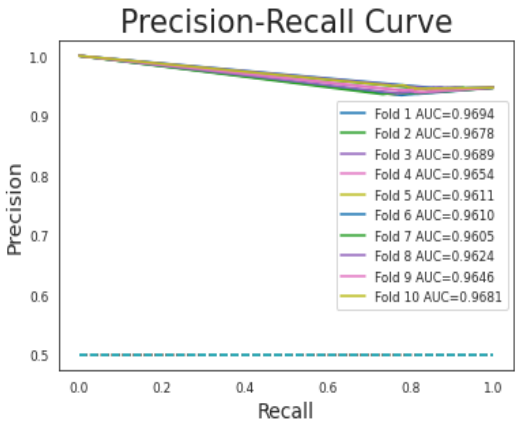


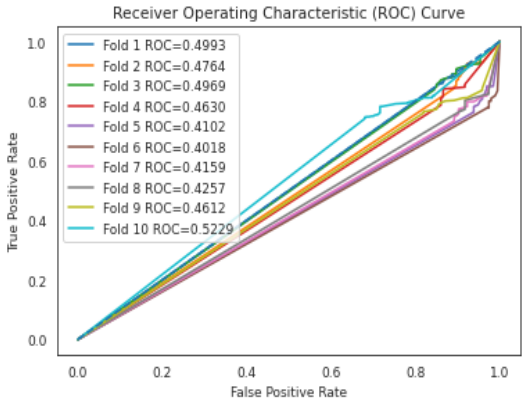


Decision Tree (DT)

	0	1	accuracy	macro avg	weighted avg
precision	0.035256	0.941639	0.752748	0.488447	0.893332
recall	0.134433	0.787559	0.752748	0.460996	0.752748
f1-score	0.055377	0.857195	0.752748	0.456286	0.814461
support	194.300000	3451.400000	0.752748	3645.700000	3645.700000

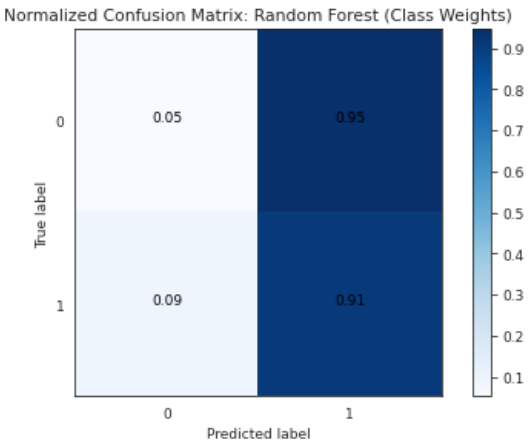


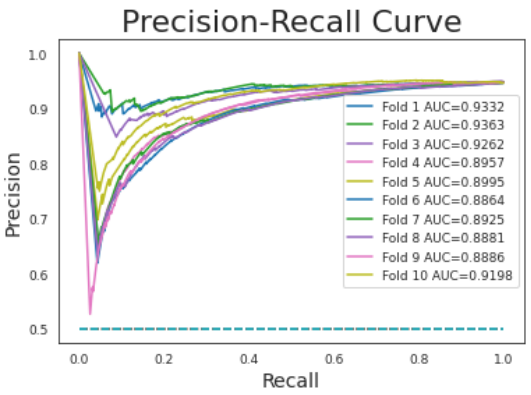


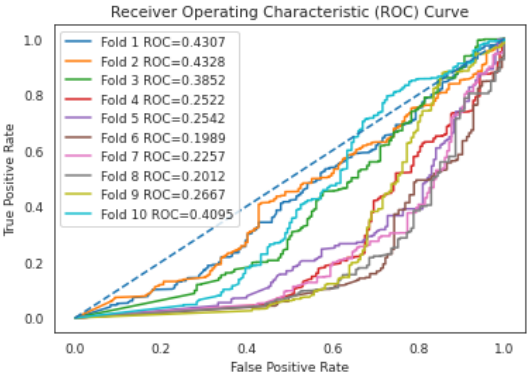


Random Forest (RF)

	0	1	accuracy	macro avg	weighted avg
precision	0.040169	0.944524	0.862191	0.492347	0.896324
recall	0.054076	0.907685	0.862191	0.480880	0.862191
f1-score	0.040921	0.925300	0.862191	0.483111	0.878166
support	194.300000	3451.400000	0.862191	3645.700000	3645.700000







Light Gradient Boosting Machine (LightGBM)

	0	1	accuracy	macro avg	weighted avg
precision	0.048798	0.944805	0.701836	0.496802	0.897051
recall	0.246135	0.727496	0.701836	0.486816	0.701836
f1-score	0.081006	0.821030	0.701836	0.451018	0.781589
support	194.300000	3451.400000	0.701836	3645.700000	3645.700000