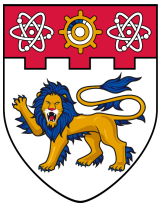
SC4000

Machine Learning



NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

Group 16

Project Report American Express - Default Prediction

Group Members:

Tan Jing Han Chad [U2222125D]
Michael Santoso [U2221525L]
Ng Jun Yu [U2222345E]
Christopher Angelo [U2220148L]
Nguyen Tien Dat (John) [U2220949L]

Table of Contents

1. Introduction

1.1. Project Description

Credit cards play an important role in modern life, offering both convenience and flexibility when making everyday purchases. Whether one is at a restaurant or making online purchases, the actual or digital credit cards can get it done. While they allow consumers to spend without carrying cash and even adopt a "buy now pay later" policy, the responsibility of managing the risk falls on card issuers. One of the biggest questions for lenders, especially commercial banks, is determining whether a customer is likely to repay what they borrow.

To accurately predict credit default is essential for financial institutions, as this facilitates their ability to make more informed lending decisions and manage risk more effectively. In this project, we will focus on applying machine learning techniques to develop and optimize various models that can predict credit default using the large and complex dataset provided by American Express. The dataset includes anonymized customer profiles along with behavioral data collected over time, which creates both an opportunity and a challenge for model development.

While the goal of our project is simply to predict if a customer will default in the future, this presents the opportunity to build a model that can improve on existing solutions, helping lenders make better decisions and offering a smoother experience for customers seeking credit.

1.2. Evaluation Metrics

The evaluation metric, M, for this competition is the mean of two measures of rank ordering: Normalized Gini Coefficient, G, and default rate captured at 4%, D.

$$M=0.5 \cdot (G+D)$$

The default rate captured at 4% is the percentage of the positive labels (defaults) captured within the highest-ranked 4% of the predictions, and represents a Sensitivity/Recall statistic.

For both of the sub-metrics G and D, the negative labels are given a weight of 20 to adjust for downsampling.

This metric has a maximum value of 1.0.

2. Exploratory Data Analysis

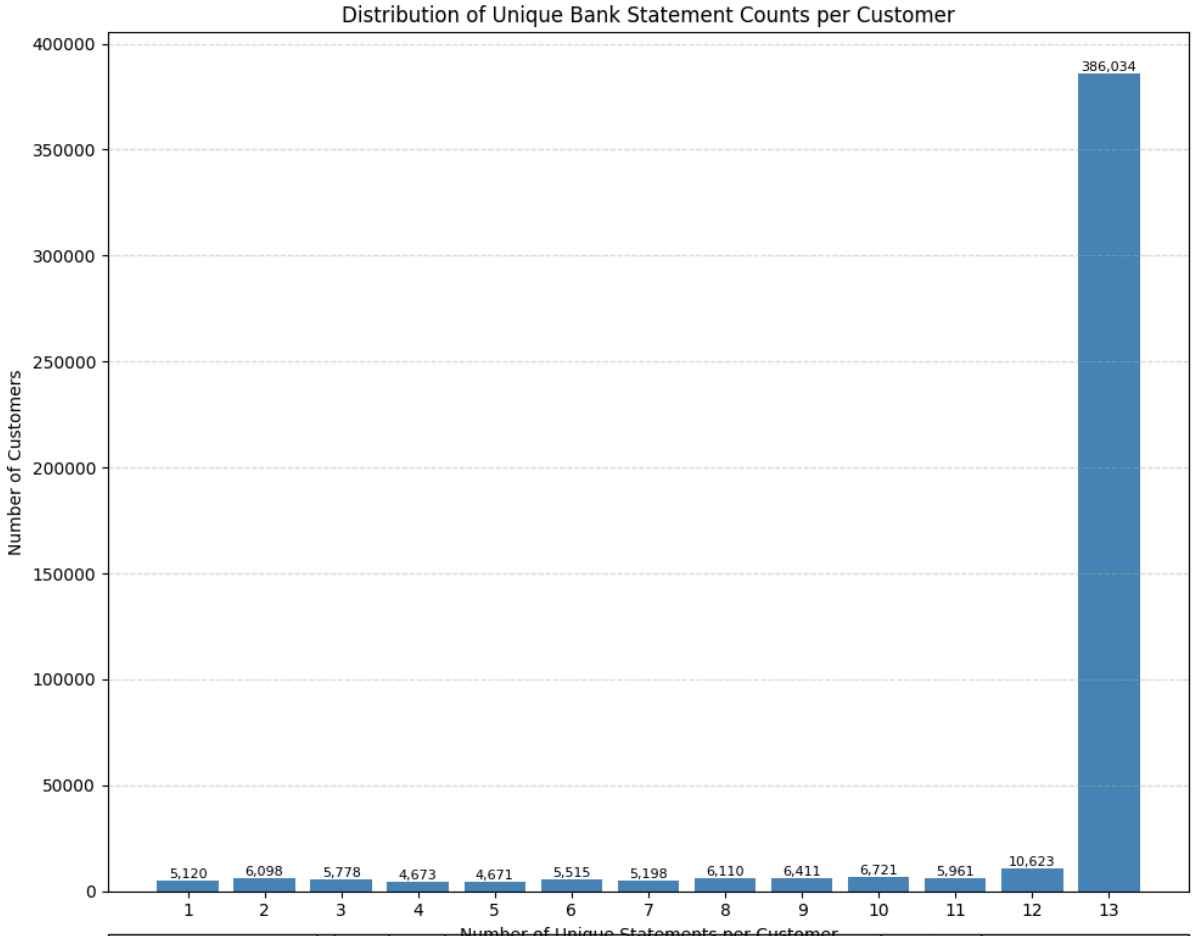
2.1 Data handling

The competition dataset is large, with the train_data.csv file spanning 16GB and containing multiple statement dates per customer, while the test_data.csv file is even bigger at 33GB. Loading the raw CSV files directly into memory is impractical due to their size. However, we can significantly reduce the memory footprint without losing data by optimizing column data types (as detailed in Section 3) and switching to a compressed file format like Parquet.

Additionally, the dataset possibly contains undesirable variables and noise such as outlier values, rows with many missing column values etc.

2.2 Dataset Overview

For most customers, the first and last statements span about a year (12 months). Given that the graph shows that most customers have a statement up to 13 months we can conclude that the total number of statements a customer (who did not default) would be 13 per customer.



statement_count	customer_count
1	5120
2	6098
3	5778
4	4673
5	4671
6	5515
7	5198
8	6110
9	6411
10	6721
11	5961
12	10623
13	386034

Figure 1: Distribution of Unique Bank Statement Counts per Customer

The target classes exhibit significant imbalance, rendering accuracy an inadequate performance metric. This class imbalance further underscores the need to analyze these categories independently for feature relevance. Additionally, we observe that some columns contain a substantial proportion of missing values.

3. Data preparation

Preparing the data was one of the most important steps in building a strong predictive model for this project. The dataset provided by American Express contains a large number of features and spans multiple time periods for each customer, so it was crucial to structure the data in a way that preserved useful information while making it manageable for modeling.

The dataset chosen is using the already integer cleaned dataset, where float data types are transformed to int data types if possible. Cleaned by *Raddar*, this dataset can improve memory efficiency, transform discrete float values (e.g. 1.0, 3.0) into normal integer values, and improve compatibility with categorical features encoding.

3.1 Data Pre-processing Techniques

This section outlines the data preprocessing methodologies implemented across successive stages of our data preparation pipeline, including the justification for each procedural step.

3.2 Feature Separation

To start, the features were divided into two categories: categorical and numerical. The distinction was based on an initial review of the variables. Features that appeared to be flags, labels, or encoded classifications were treated as categorical, while the other features which are mostly continuous or count-like variables, were handled as numerical. There are two other columns not included as features, which are customer_ID and S_2 (the date of transaction). Separating the features early on helped streamline the preprocessing process and allowed for more targeted feature engineering later.

3.3 Feature Engineering

Since each customer has a sequence of records over time, one of the main goals was to convert these into meaningful summaries. For each numerical feature, we created a set of aggregate values which included the mean, max, min, standard deviations, and last value. These parameters are calculated across the customer's full history. This gave us a snapshot of each customer's behavior without losing important variation.

In addition, the categorical features are also transformed using aggregation, producing count, last, unique (number of unique values). These values are used to help predict behaviour of customers.

3.4 Incorporating Time

Since the data provided is inherently time-based, it was important to reflect this unique aspect in the features. One approach we used was to extract the last available value for each feature. The most recent information about a customer, such as their latest transaction amount or credit usage, tends to be particularly useful in predicting whether they might default soon. Thus, the last value is taken and recorded for both numerical and categorical features.

4. Model training

We experimented with multiple models and architectures to identify the best-performing approach. After extensive testing and tuning, the results demonstrated varying levels of performance across different algorithms. Key findings and comparative metrics are summarized below, highlighting the most effective models for this task.

4.1 LightGBM with StratifiedKFold

In the Kaggle American Express Default Prediction Competition, the goal is to predict whether a customer will default on their payments. This is a form of binary classification problem and is highly imbalanced since the possibility of defaults is lower than non-defaults.

We first trained a baseline model using LightGBM, a gradient boosting framework that is fast, efficient and is capable of handling large datasets and missing values well. It can handle imbalanced datasets well in our context, using the scale_pos_weight parameter or other evaluation metrics.

Since the dataset is highly imbalanced, StratifiedKFold ensures that each training fold has the same proportion of defaults and non-defaults as the initial dataset. This will prevent biases against defaults since they are more rare.

4.1.1 Key Hyperparameters

StratifiedKFold() is used to train and predict the validation and test datasets. After testing different configurations, we found out that using 5 fold provided the best results. Below are some of the parameters used in the model:

Hyperparameter	Value	Rationale
n_estimators	**1000**	A higher number of trees can improve performance if paired with a lower learning rate.This allows the model to learn gradually and avoid overfitting. Since the learning rate is low (0.03), a high n-estimator (1000) compensates by giving the model sufficient capacity to learn over time.
learning_rate	**0.03**	A low learning rate helps the model generalize better and reduces the risk of overfitting, especially on complex or noisy data. It requires a high number of trees (1000) to achieve optimal performance.
num_leaves	**50**	A moderate number of leaves balances between model flexibility and overfitting. Too many leaves can lead to overfitting while too few may cause underfitting.
reg_alpha	**2**	Reduces overfitting by penalizing large weights. An alpha value of 2 suggests the model is strongly regularized. This is particularly useful when the dataset is large and complex, as in the Amex dataset.
min_child_samples	**2000**	The large value of 2000 heavily restricts the model from making splits that create small, potentially noisy leaves.

		It forces the model to only learn patterns that are present in larger groups of samples to prevent over-fitting, in large datasets like the Amex dataset.

Figure 2: Hyperparameters for LightGBM with StratifiedKFold and the rationale behind each one of them

Each fold includes plots of binary log loss to track the training progress. Since accuracy is not a supported metric for LGBMClassfier(), it's replaced with AUC (Area Under Curve).

4.1.2 LightGBM Plot

This is the plot for the training and validation binary log loss against the training iterations for LightGBM classifier.

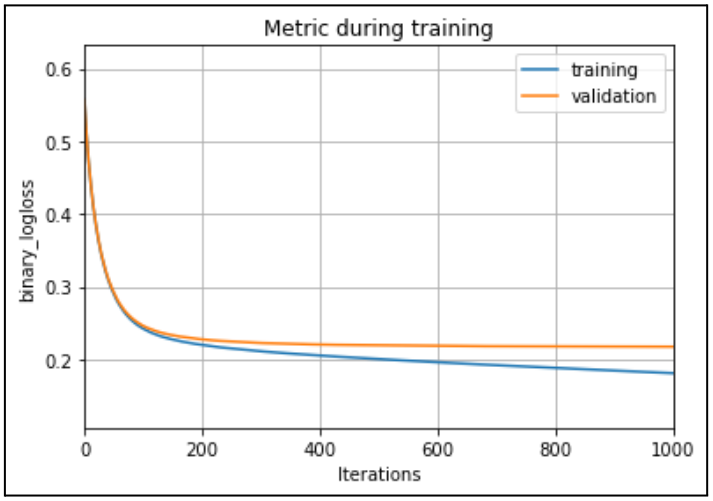


Figure 3: binary_logloss against number of iteration during training

X-axis (Number of Iterations)

- This axis represents the number of training steps
- As the model trains, it updates its parameters to allow the data to fit better

Y-axis (binary_logloss)

- This axis measures how well the predicted probabilities can match the actual labels
- A lower log loss results in a better performance

Number of iterations	Observed Behaviour
0 ~ 100 iterations:	Initially, both the training and validation curves dip sharply, showing that the model is learning rapidly from the dataset.
100 ~ 200 iterations:	The training and validation curves start to flatten, which highlights diminishing improvements to the learning rate as training progresses.
200 ~ iterations:	The training loss curve is slightly lower than the validation loss curve as the model is more adapted to the training data and "fits" it better, compared to validation data.

Figure 4: Our observations on the behaviour of the binary log loss curve given a range of iterations

The training loss curve is slightly lower than the validation loss curve as the model is more adapted to the training data and "fits" it better, compared to validation data.

Since both curves are generally close to one another, there is no overfitting problem and there is decent generalization on the training data.

4.1.3 Quantitative Results

	Precision	Recall	F1-Score
0	**0.93**	**0.94**	**0.94**
1	**0.82**	**0.81**	**0.81**
Macro Avg	**0.88**	**0.87**	**0.87**
Weighted Avg	**0.90**	**0.90**	**0.90**

Figure 5: Quantitative results after training (e.g precision, f1-score etc)

Model	Train F1-Score 0	Train F1-Score 1	Train Accuracy	Train M Score
LightGBM with StratifiedKFold	0.94	0.81	90%	0.7920

Figure 6: Final results after training and normalization (e.g train score and accuracy score) for LightGBM with StratifiedKFold

The model achieves an F1-Score 0 of 0.94 and F1-Score 1 of 0.81. Hence, it's more confident in predicting non-default customers (Class 0) compared to defaulting customers (Class 1).

With a high train accuracy of 90%, the model is able to fit the data well and generalize well on it as well.

Overall, the baseline LightGBM model provided a strong starting point with an M score of 0.7920, serving as a benchmark for further improvements.

4.1.4 Improvements

The discrepancy in F1-score 0 (0.94) and F1-score 1 (0.81) indicates that the model was imbalanced in their predictive capabilities for defaulting and non-defaulting customers. Attempting to raise the F1-score 1 to match F1-score 0 forms the basis of our attempts at improving the model.

One attempted method for raising the F1-score 1 to match F1-score 0 is threshold optimization, where we try to adjust the classification threshold to balance recall (detect defaulters) and precision (avoiding false alarms) for real-world applicability. By adjusting this threshold, we can tweak our model to be more sensitive (higher recall) or more selective (higher precision) depending on the context of the situation and industry.

If the cost of overlooking a defaulter is high, we might lower the threshold to classify more customers as potential defaulters, improving recall. If the cost of false positives (wrongly flagging reliable clients) is high, we can raise the threshold to improve precision.

4.2.1 Ensemble learning

To push our model performance further, we experimented using an ensemble learning approach. Rather than relying on a single model, ensemble learning aggregates the predictions of multiple models, allowing for reduction of variance and potentially correcting of individual model weaknesses. The idea is that each model may capture slightly different aspects of the data, and by blending them, whether through averaging or more complex stacking, we end up with more stable and accurate results.

We selected ensemble learning as our final modeling approach for several key reasons. First, the ensemble consistently delivered higher predictive accuracy than a single LightGBM with StratifiedKFold, particularly in identifying borderline default cases. Additionally, blending multiple models improved generalization by reducing overfitting and smoothing out inconsistencies in individual predictions. By incorporating different LightGBM variants, such as DART, we also increased robustness to model bias and minimized reliance on any single set of hyperparameters. Finally, our review of top Kaggle leaderboard submissions revealed that most leading entries used ensemble or hybrid models, giving us further confidence that this strategy would make our solution more competitive.

4.3.1 Ensemble learning (Combining LightGBM with MLP)

We first attempted combining our base LightGBM model with a Multi-Layer Perceptron (MLP) — neural network architecture known to capture non-linear relationships and intricate interactions between features differently than tree-based methods. Our hypothesis was that the MLP might identify complementary predictive patterns specifically beneficial for the underrepresented defaulters class, potentially improving recall or at least reducing bias toward the majority non-defaulters class.

4.3.2 Results

Model	Train F1-Score 0	Train F1-Score 1	Train Accuracy	Train M Score
Ensemble Learning (LightGBM with MLP)	0.94	0.83	91%	0.7890

Figure 7: Final results after training and normalization (e.g train score and accuracy score) for Ensemble Learning (LightGBM with MLP)

This new model achieved an F1-Score 0 of 0.94 and F1-Score 1 of 0.83 with a train accuracy of 91%, a 0.02 increase in F1-score 1 and a 0.01 increase in train accuracy. Making this model more confident in predicting defaulting customers with greater accuracy.

However, there was a counterintuitive result of the M score decreasing from 0.792 to 0.789 despite closing the gap between the F1-score 0 and F1-score 1, and increasing the train accuracy. This highlights the difference between threshold-based metrics like F1 and accuracy and the ranking-based nature of the M-score.

In the context of this competition, we learnt that what matters most is not just predicting defaults correctly, but ranking the riskiest 4% better. The drop in M score suggests that, despite better label predictions, the model became less confident in its ranking, possibly due to calibration loss introduced by the MLP.

4.4.1 Ensemble Learning (LightGBM Variants)

To achieve further improvements and address limitations observed in prior models, we adopted an ensemble approach, inspired by top-performing methods in the Amex Default Prediction Competition and refined their techniques to generate more robust predictions.

Specifically, we utilized LightGBM with DART (Dropouts meet Multiple Additive Regression Trees) boosting as our primary ensemble method and trained multiple LightGBM models using K-fold Cross Validation. The generated predictions using each of the 5 trained LightGBM models are then averaged out providing the final prediction for each test sample.

4.4.1 Key Hyperparameters

Our final ensemble model was built on a carefully selected set of hyperparameters, each chosen to maximize ranking performance. After experimenting with various configurations, we identified the following optimal settings.

Hyperparameter	Value	Rationale
Boosting Type	**DART**	DART introduces dropout regularization by randomly dropping trees during training, which helps prevent overfitting and improves model robustness.
No. of Leaves	**100**	Using 100 leaves allows the model to capture complex patterns while balancing expressiveness and the risk of overfitting.
Learning rate	**0.01**	A low learning rate ensures gradual learning, leading to more stable and accurate results over many boosting rounds.
Feature Fraction	**0.02**	By using only 2% of features per boosting round, the model reduces overfitting and focuses on the most informative features.
Bagging Fraction	**0.5**	Sampling half the training data in each iteration introduces randomness, enhancing generalization and reducing variance.
Regularization	**2**	L2 regularization with a value of 2 penalizes large weights, helping the model avoid fitting noise and improving generalization.
Early stopping rounds	**100**	Training stops if validation performance doesn't improve for 100 rounds, which saves computation and prevents overfitting.

Figure 8: Hyperparameters for LightGBM with MLP and the rationale behind each one of them

We chose these specific hyperparameter values because this configuration consistently delivered the best results on our validation data and public leaderboard.

4.4.1 Final Model Evaluation and Results

To ensure robust performance, we ran the LGBM model 15 times, each time allowing for different random initializations and data shuffling. This approach introduces subtle variations in model outcomes, even with identical hyperparameter settings. From these 15 runs, we selected the top three variants based on their public leaderboard scores.

Rank	Leaderboard Score
1	0.79971
2	0.79967
3	0.79944

Figure 9: Public Leaderboard Scores of Top 3 LGBM Variants

To further enhance performance, we created a weighted ensemble of these top three models, assigning greater weight to those with higher individual scores. This ensemble achieved a final public leaderboard score of **0.79986**, outperforming any single model.

Model	Train F1-Score 0	Train F1-Score 1	Train Accuracy	Train M Score
Ensemble Learning (combined LightGBM variants)	0.945	0.84	92%	0.79986

Figure 10: Final results after training and normalization (e.g train score and accuracy score) for Ensemble Learning (LightGBM with MLP)

This final LightGBM ensemble achieved stable improvements, improving F1-score 0 to 0.945, F1-Score 1 to 0.84 and train accuracy to 92%, achieving a final M-score of 0.79986. This improvement can be attributed primarily to the combination of diverse LightGBM variants. The ensemble's emphasis on stochastic regularization allowed the model to better capture subtle predictive patterns in the data while maintaining robust generalization.

4.4.1 Leaderboard Ranking

With a final public leaderboard score of **0.79986**, our submission ranked 699th out of 4,874 teams, placing us in the **top 14.34%** of all participants.

$$Percentage\ Rank = \frac{699}{4874} \times 100 = 14.34\%$$

For context, the top-ranked model achieved a score of 0.80203, meaning our ensemble is behind the leader by just 0.00217.

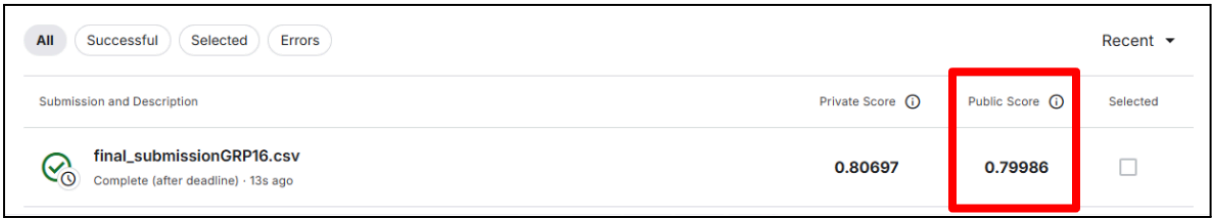


Figure 11: The public score of our submission on Kaggle public leaderboard

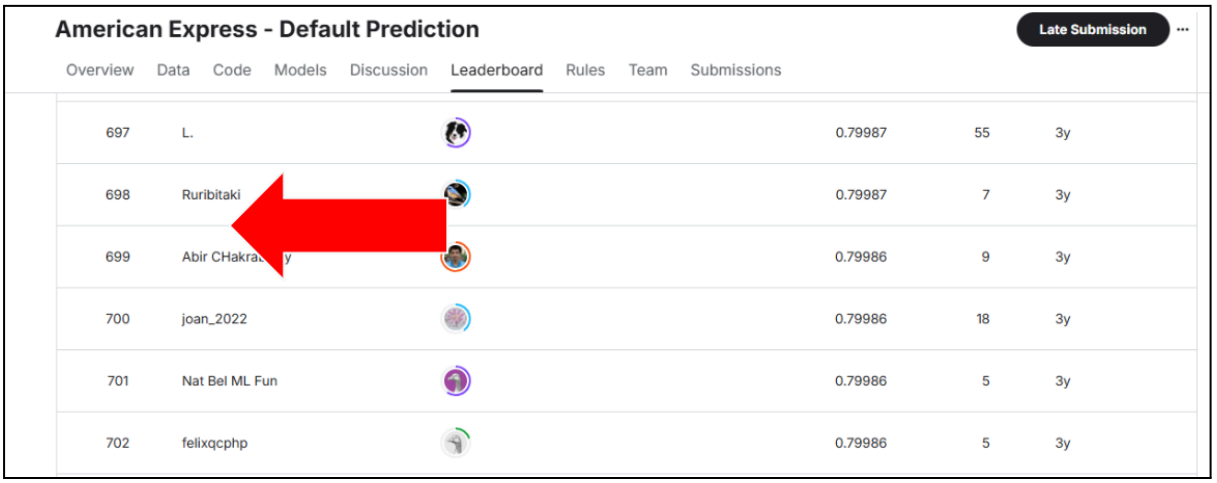


Figure 12: Our ranking on the Kaggle public leaderboard

5. Conclusion

In this project, we explored a few machine learning approaches to tackle the American Express Default prediction kaggle competition. We used LightGBM, combined with StratifiedKfold cross-validation and it emerged as a decent baseline model. It managed to effectively handle the class imbalance and the large scale data while achieving an M-score of 0.7920 and an F1-score of 0.81 for defaulters (Class1). The training data showed a stable convergence without any overfitting.

To further enhance performance, we attempted ensemble learning to take advantage of the strengths of each individual model while correcting their weaknesses, yielding more robust final predictions.

First we attempted a LightGBM with MLP blend which resulted in a better label balance with a F1-score of 0.83 for defaulters (Class 1) but a lower M score of 0.7890. This counter-intuitive result demonstrated to us that raising F1-score 1 via threshold shifts might improve traditional metrics but not the more ranking sensitive M score, underlining the need to optimise our model more specifically for the competition's objectives and not simply generic accuracy. Adding an MLP had slightly bolstered recall but diluted the LightGBM's calibrated probabilities, overall hurting the 4% ranking rate.

Recognizing LightGBM's strong ranking capabilities, we focused on homogeneous ensembling by training five LightGBM-DART models using different cross-validation folds and averaging their predictions. This approach produced our best results, with a Train F1-score of 0.945 for Class 0, 0.84 for Class 1, a training accuracy of 92%, and an M-score of 0.79986. Our final submission ranked in the top 14.34% on the public leaderboard.

Overall, we had fun exploring various algorithms and implementing different machine learning models introduced during lectures. For future work, we suggest exploring advanced sampling techniques such as SMOTE or ADASYN to address class imbalance, conducting deeper hyperparameter optimization, and investigating alternative model architectures to further enhance performance.

6. References

- ResearchGate. (n.d.). *LightGBM model computing process* [Figure]. Retrieved April 23, 2025, from https://www.researchgate.net/figure/LightGBM-model-computing-process_fig3_359123597
- GeeksforGeeks. (2022, September 19). *Cross-validation and hyperparameter tuning of LightGBM model*. Retrieved April 23, 2025, from https://www.geeksforgeeks.org/cross-validation-and-hyperparameter-tuning-of-lightgbm-model/
- fintech-quagga-group. (n.d.). *American Express Default Prediction* [GitHub repository]. GitHub. Retrieved April 23, 2025, from https://github.com/fintech-quagga-group/american-express-default-prediction/tree/main
- IBM. (n.d.). *Ensemble learning*. IBM Think. Retrieved April 23, 2025, from https://www.ibm.com/think/topics/ensemble-learning
- Data Science. (2018, November 15). *Designing your neural networks*. Medium. Retrieved April 23, 2025, from https://medium.com/data-science/designing-your-neural-networks-a5e4617027ed
- GitHub. (n.d.). *Managing large files*. GitHub Docs. Retrieved April 23, 2025, from https://docs.github.com/en/repositories/working-with-files/managing-large-files
- jxzly. (n.d.). *Kaggle American Express Default Prediction – 1st solution* [GitHub repository]. GitHub. Retrieved April 23, 2025, from https://github.com/jxzly/Kaggle-American-Express-Default-Prediction-1st-solution
- isiriwithanawasam. (n.d.). *Amex default prediction* [Kaggle notebook]. Kaggle. Retrieved April 23, 2025, from https://www.kaggle.com/code/isiriwithanawasam/amex-default-prediction
- ragnar123. (n.d.). *Amex LGBM Dart CV 0.7977* [Kaggle notebook]. Kaggle. Retrieved April 23, 2025, from https://www.kaggle.com/code/ragnar123/amex-lgbm-dart-cv-0-7977/output
- slowlearnermack. (n.d.). *Amex LGBM Dart CV 0.7963 (Improved)* [Kaggle notebook]. Kaggle. Retrieved April 23, 2025, from https://www.kaggle.com/code/slowlearnermack/amex-lgbm-dart-cv-0-7963-improved/output
- raddar. (n.d.). *Amex data - integer dtypes parquet format* [Kaggle dataset]. Kaggle. Retrieved April 23, 2025, from https://www.kaggle.com/datasets/raddar/amex-data-integer-dtypes-parquet-format/data