Home Loan Default - Risk Management

Team project Id: **1198**
Project ID: **1006**
Project Team Members
1. **Lavanya Kancharla**
2. **Ruhi Poul**
3. **Pramod Naik**
4. **Pooja KS**
5. **Suraj Sawane**

Problem statement:
Building a model to predict how capable each applicant is of repaying a loan, so that sanctioning loan only for the applicants who are likely to repay the loan.

Data Description and Overview:
The data is provided by Home Credit, a service dedicated to provided lines of credit (loans) to the unbanked population.
There are 7 different sources of data:
application train: The main training data with information about each loan application at Home Credit. Every loan has its own row and is identified by the feature SK_ID_CURR. The training application data comes with the TARGET indicating 0: the loan was repaid or 1: the loan was not repaid. Here we will use only the Training data.
bureau: In this dataset it consists of data concerning client's previous credits from other financial institutions. Each previous credit has its own row in bureau, but one loan in the application data can have multiple previous credits.
bureau balance: It consists of monthly data about the previous credits in bureau. Each row is one month of a previous credit, and a single previous credit can have multiple rows, one for each month of the credit length.
previous application: The data of previous applications for loans at Home Credit of clients who have loans in the application data. Each current loan in the application data can have multiple previous loans. Each previous application has one row and is identified by the feature SK_ID_PREV**.**
POS_CASH_BALANCE: It consists of monthly data about previous point of sale or cash loans clients have had with Home Credit. Each row is one month of a previous point of sale or cash loan, and a single previous loan can have many rows.
credit_card_balance: The monthly data about previous credit cards clients have had with Home Credit. Each row is one month of a credit card balance, and a single credit card can have many rows.
installments_payment: The data of payment history for previous loans at Home Credit. There is one row for every made payment and one row for every missed payment.
Mapping to an Machine Learning Problem:
For the given applicant data we need to predict if they are capable to repay the loan or not. Since we have two classes in the target label, we will pose this problem as a Binary classification problem.

Data Preparation As the first step, the necessary libraries and the datasets are imported. Since there are more than one files, all need to be imported before looking at the feature types and number of rows/columns in each file. For the main training data, there are 307511 total samples (each row a separate loan) with 122 features of types 41 integer, 65 float and 16 object datatypes. Out of these, the feature (SK_ID_CURR) serves as the index and TARGET is the response feature to be predicted.

Exploratory Data Analysis After data importing, we can investigate the data to check data quality and find trends. On plotting the distribution of the predictor variable, we can see in Figure 2 that the dataset is imbalanced with the number of samples where loan is repaid (282,686) more than 10 times the number of samples where loan is defaulted (24,825).

Feature Engineering After exploring the data distributions, we can conduct feature engineering to prepare the data for model training. This includes operations like replacing outliers, imputing missing values, one-hot encoding categorical variables, and rescaling the data. The process of replacing outliers involves removing values from data which are greater than three standard deviations from the mean. Since categorical variables cannot be directly interpreted by most classifiers, they need to be encoded as numbers. This can be done by label encoding or one-hot encoding. Label encoding assigns each category in a feature with an integer without creating new columns. One-hot encoding10 creates a new column for each category where each observation is assigned as 1, while the other category columns are assigned value of 0. It is preferred in this project since it is possible that the label numbers in the label encoding are wrongly interpreted by the model as holding some significance, while one-hot encoding does not have that issue. For tackling missing values, a two-step approach is followed. The features which have more than 60% data missing are removed, while the remaining features have data imputed, i.e., categorical features are filled with the most frequent column, and other features are filled with the median value. Rescaling of the features involves transforming each feature to a range of 0-1.

Logistic regression is a supervised classification algorithm that uses the sigmoid function (σ) (equation 1B) to convert the linear regression equation (t) (equation 1A) formed by the predictor features (X) into binary classes, 0 or 1 using decision thresholds. $t = \beta 0 + \beta 1X1 + \beta 2X2 + \cdots + \beta nXn$ (1A) $\sigma(t) = 1\ 1{+}e{-}t$ (1B) The model from the Scikit-learn14 package was implemented, and when applied with the default hyperparameters (entire dataset), it gave an accuracy of 0.67

Random Forest is an ensemble tree-based learning algorithm that uses multiple decision trees from randomly selected subsets of training data for classification. It uses averaging from the ensembles to build class predictions, while improving the predictive accuracy and control over-fitting. The model from the Scikit-learn package was implemented, and when applied with the default hyperparameters, it gave an accuracy of 0.8451, F1-score of 0.2208 and AUC of 0.6670.

Decision Tree is an algorithm where the data is iteratively split into partitions using all the features of the dataset for classification. While the accuracy increases with more splits, it can lead to overfitting when used without cross-validation. The model from the Scikit-learn package was implemented, and when applied with the default hyperparameters, it gave an accuracy of 0.85

Gaussian Naïve Bayes is a supervised classification algorithm based on applying Bayes' theorem with the "naive" assumption of conditional independence between every pair of features and the likelihood is assumed to be Gaussian. The model from the Scikit-learn package was implemented, and when applied with the default hyperparameters, it gave an accuracy of 0.64

XGBoost is an optimized distributed gradient boosting library, which uses Gradient Boosting algorithm for efficient classification. XGBoost provides parallel tree boosting to build multiple weak prediction decision tree models to perform fast and accurate predicition. The model from the XGBoost package was implemented, and when applied with the default hyperparameters, it gave an accuracy of 0.80

Gradient Boosting is a generalization of boosting to arbitrary differentiable loss functions. It uses multiple weak learners (regression trees) additively in a forward stage-wise fashion and generalizes them by allowing optimization of a differentiable loss function. The model from the Scikitlearn package was implemented, and when applied with the default hyperparameters, it gave an accuracy of 0.71

Conclusion and future work

A machine learning-based classification tool to predict the loan default risk which uses more features than just the traditional credit history can be of great help for both, potential borrowers, and the lending institutions. This project attempts to create a complete end-to-end workflow for building a binary classifier and includes methods like automated feature engineering for connecting relational databases, comparison of different classifiers on imbalanced data, and hyperparameter tuning using Bayesian optimization. The process was able to achieve a highest ROC AUC score of 0.7836 on the test data from random search hyperparameter tuning, while the base case AUC score was of 0.7454 on the test data. Thus, the final model could adequately predict cases when the loan may be defaulted. However, I was not able to exceed the higher end benchmark of Kaggle leader board (AUC: 0.817). In the future, I would like to improve the model further to get a higher AUC score, possibly by creating more domain-knowledge-based features, exploring more classification models, and conducting greater iterations for hyperparameter optimization using cloud computing resources.