Infosys | Springboard

Internship - Project Report

Home Credit Default Risk Recognition

Submitted By

Team Members List

1. Sujal Shinde
2. Rakshana
3. Hemanth
4. Amisha
5. Pari

INDEX

1.0 Project background

An important fraction of the population finds it difficult to get their home loans approved due to insufficient or absent credit history. This prevents them to buy their own dream homes and at times even forces them to rely on other sources of money which may be unreliable and have exorbitant interest rates. Conversely, it is a major challenge for banks and other finance lending agencies to decide for which candidates to approve housing loans. The credit history is not always a sufficient tool for decisions, since it is possible that those borrowers with a long credit history can still default on the loan and some people with a good chance of loan repayment may simply not have a sufficiently long credit history.

A number of recent researchers[2,3,4] have applied machine learning to predict the loan default risk. This is important since 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. Hence, an attempt is made to train a classifier using machine learning techniques on the given dataset to help determine the risk of loan default.

2.0 Problem statement and solution strategy

The problem and associated data has been provided by Home Call Credit Group[1], and the problem can be summarized as, *"A binary classification problem where the inputs are various features describing the financial and behavioral history of the loan applicants, in order to predict whether the loan will be repaid or defaulted."*

The solution workflow has been divided into five parts (as shown in the Jupyter notebook): i) Data Preparation ii) Exploratory Data Analysis iii) Feature Engineering iv) Classifier Model: Training, Prediction, Comparison, v) Hyperparameter Tuning. Initially, the datasets are imported, the missing values are resolved, and categorical features are encoded. Feature engineering is used to connect the data across the different relational databases and make new features. After the exploratory data analysis and feature engineering, the available features are used for training the classification model. Different classification models like logistic regression, random forest, and gradient boosting with various libraries are trained for selecting the best one for this problem. Thereafter, the model is refined using K-fold crossvalidation and hyperparameter tuning with methods like grid search, random search, or Bayesian optimization.

3.0 Datasets and inputs

The dataset files are provided on the Kaggle website (https://www.kaggle.com/c/home-creditdefault-risk/data) in the form of multiple CSV files and are free to download. The dataset files are described as per Figure 1.

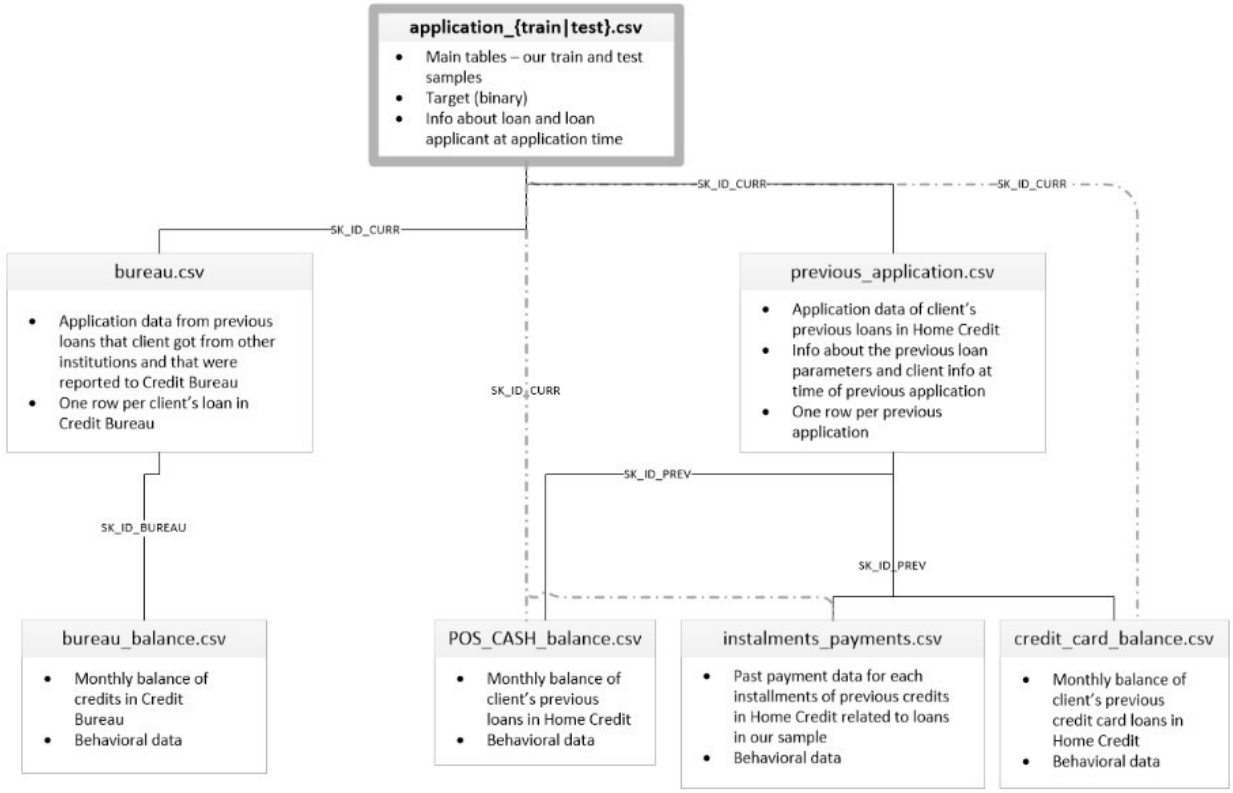

application_{train|test}.csv
Main tables – our train and test samples
Target (binary)
Info about loan and loan applicant at application time
SK_ID_CURR
SK_ID_CURR
SK_ID_CURR
SK_ID_CURR
bureau.csv
Application data from previous loans that client got from other institutions and that were reported to Credit Bureau
One row per client's loan in Credit Bureau
previous_application.csv
Application data of client's previous loans in Home Credit
Info about the previous loan parameters and client info at time of previous application
One row per previous application
SK_ID_PREV
SK_ID_BUREAU
SK_ID_PREV
bureau_balance.csv
Monthly balance of credits in Credit Bureau
Behavioral data
POS_CASH_balance.csv
Monthly balance of client's previous loans in Home Credit
Behavioral data
instalments_payments.csv
Past payment data for each installments of previous credits in Home Credit related to loans in our sample
Behavioral data
credit_card_balance.csv
Monthly balance of client's previous credit card loans in Home Credit
Behavioral data

Figure 1- Description and connectivity of the Home Credit Default Risk dataset[1]

As seen in Figure 1, the file *application_{train|test}.csv* contains the main table containing the training dataset (307511 samples) and test dataset (48744 samples), with each row representing one loan identified by the feature SK_ID_CURR. The training set contains the variable TARGET with binary values (0: the loan was repaid or 1: the loan was not repaid). Since, we do not have access to the labeled version of the test dataset, I did not use it, instead I split part of the training dataset to use as the test data. The other datasets in Figure 1, are bureau.csv, bureau_balance.csv, previous_application.csv, POS_CASH_balance.csv, instalments_payment.csv, credit_card_balance.csv. Since there are so many input files available, they need to be connected to the main application training file using extract, transform, load (ETL) processes which can deal with relational databases. This will connect each parentchild database using a common index, and the additional features may show some correlations with the target feature. Thus, the larger number of input features and training samples will allow better identification of the important factors for the credit default risk classification model.

4.0 Benchmark model and evaluation metrics

A receiver operating characteristic (ROC) curve can summarize the performance of a binary classification model on the positive class where the x-axis shows False Positive Rate and the y-axis shows the True Positive Rate. A ROC curve does not have bias towards the majority or minority class since it uses the actual predicted probability, instead of the probability class, making it favorable when using imbalanced data with equal importance for the both classes[5]. The area under the ROC curve can be calculated to find a score between 0 and 1 for a classifier for all threshold values, called the ROC area under curve or AUC. Since the AUC was also used as the evaluation metric in the Kaggle competition, it has been used as the primary evaluation metric, but other metrics like accuracy and F1-score are also calculated for comparing the different models. The accuracy[6] is defined as number of correct predictions made, divided by the total number of predictions and does not work well for imbalanced datasets. The F1 score[6] is a metric which is calculated by taking the harmonic mean of the precision ($\frac{True\ Positive}{True\ Positive+False\ Positive}$) and the recall ($\frac{True\ Positive}{True\ Positive+False\ Negative}$).

Since this was originally a Kaggle competition, the benchmark model for comparison was the highest leaderboard score (AUC score: 0.817)[1]. However, since I had limited domain knowledge to create more features, and since I did not have access to the actual test dataset, my best results (test data AUC: 0.7860) did not exceed this higher-end benchmark. On the lower end, as a sanity check, I trained a base case model using logistic regression with only the main training dataset as a benchmark to improve on, which gave an AUC of 0.7454 on my test dataset.

5.0 Project design and solution

The project is implemented using different libraries in Python3 in a Jupyter Notebook[7]. The notebook provides a complete workflow for building a binary classifier and includes techniques like automated feature engineering for connecting relational databases, comparison of different classifiers using various libraries (XGBoost[8] and LightGBM[9]) on imbalanced data, and hyperparameter tuning using Bayesian optimization. These steps help in improving the classification results, and the model with the best results can be finally selected to use as a credit default risk classification model. The project has been divided into five parts which will be elaborated subsequently.

5.1 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. The dataset file names, number of rows and columns is summarized in Table 1.

Dataset name	Rows	Columns
application_train	307511	122
bureau	1716428	17
bureau_balance	27299925	3
credit_card_balance	3840312	23
installments_payments	13605401	8
pos_cash_balance	10001358	8
previous_application	1670214	37

Table 1- Dataset file names, rows, and columns

5.2 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). This imbalance should be considered during model training.

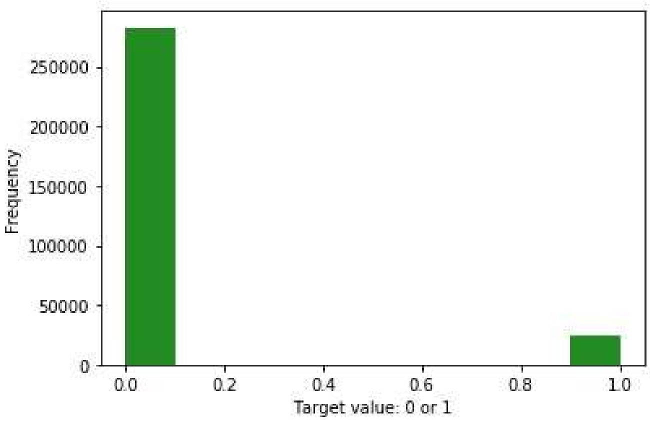


Figure 2- TARGET variable distribution: loan repaid (TARGET=0) and loan defaulted (TARGET=1) On checking for missing data, it is seen in Figure 3 that a number of features in the main dataset (application) have missing values almost 50%. The features with high fractions of missing data need to be discarded, and those with some missing values need to be imputed before training any model.

Similar checks are performed on the other datasets.

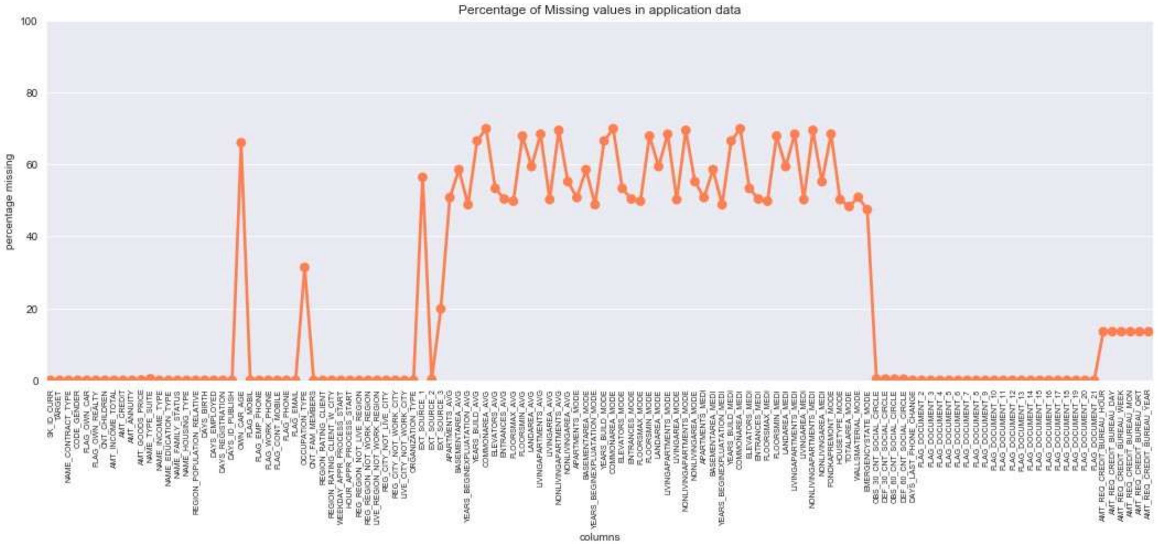


Figure 3- Percentage of missing data in main dataset

The next step entailed continuing the exploratory data analysis to see if any response features have significant differences for cases when loans are repaid as opposed to when loans are defaulted. A number of figures were created for categorical features with bar plots for each type, with each figure containing, a) the total number of categories for each response feature, b) the fraction of each category with loan defaulted. For example, for the feature income type (NAME_INCOME_TYPE) in Figure 4, we see that there are more loans taken by those who are working and they are more defaulted by those who are on maternity leave or unemployed. A comprehensive list of bar plots is available in the Jupyter notebook.

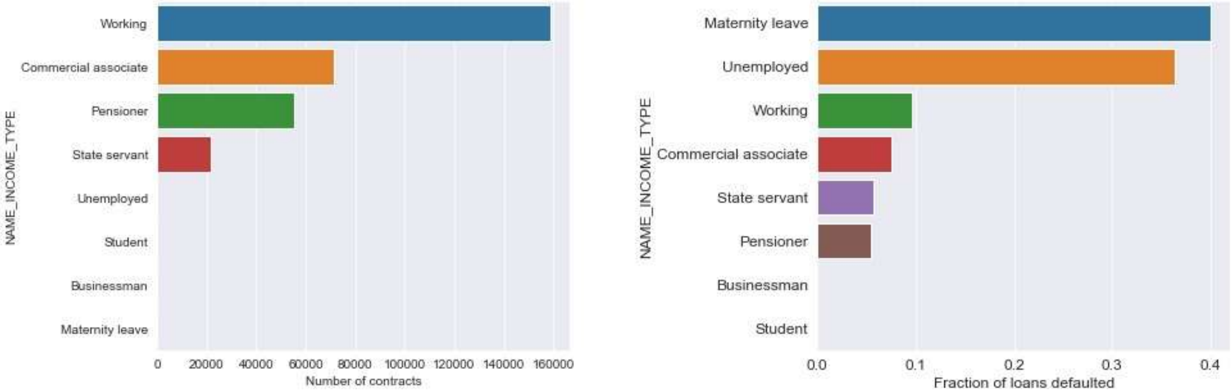


Figure 4- A) Distribution of income types for total sanctioned loans, B) Distribution of income types by fraction of loans defaulted.

For the continuous features, the distribution of a feature for the cases when loans are repaid and defaulted, were plotted to examine differences and check data quality. For example, for the feature

‘DAYS_EMPLOYED’ in Figure 5A, we can see that the distribution is hard to perceive and has some anomalous values of days employed more than 350000 days (958 years), which is impossible and needs to be corrected. On removing the outliers, and converting the days to years, we can see a more interpretable distribution in Figure 5B, where there are a greater number of loans defaulted by people who are employed for fewer years.

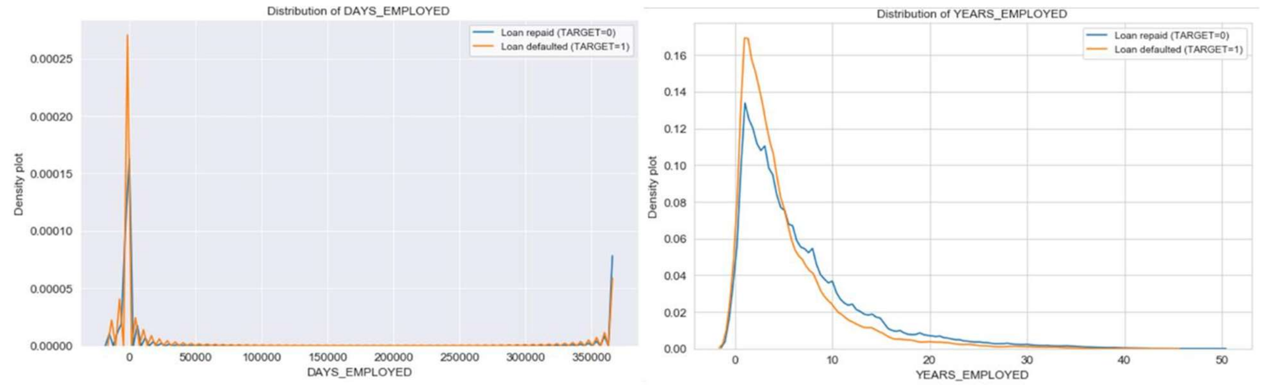


Figure 5- A) Distribution of days employed, B) Distribution of years employed (after correction)

A comprehensive list of such distribution plots is available in the Jupyter notebook. Such an analysis helps us gain an insight into the features which need to be corrected, and those which may help identify the likelihood of loan default and can turn out to be important features for model training.

5.3 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 (eg. number of days in Figure 5). 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 encoding[10] 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.

I have also applied my limited domain knowledge to create few more variables, specifically ratios accounting for the credit income %, annuity income %, credit term, and fraction of years employed. Since there are number of relational databases, we can use extract, transform, load (ETL) processes using automated feature engineering to connect the datasets. Since manual feature engineering cannot create all possible combinations of features from all the available datasets, automated feature engineering helps in building thousands of features using different operations like merge, group, sum, etc. Featuretools[11] is an open-source library for making features from related datasets using deep feature synthesis. A table/dataframe called entity is created with a unique index for all the datasets. The relationships between the datasets are defined as shown in Figure 1 using an index such as SK_ID_CURR or by creating a new index when needed. Care is taken to prevent cross-connecting relationships. This allows the multiple entities to come together to form an entity-set. Then, feature primitives are created for the datasets, where primitives are operations conducted on tables to create a feature. The two common types are aggregation and transformation. An aggregation groups together values from child dataset for each parent and then calculates a feature such as mean, min, max, or standard deviation. A transformation can be applied to one or more columns in a single table such as the difference between two columns or the absolute value of one column. Some common primitives in Featuretools are count, mean, median, trend, maximum, minimum, number of words, cumulative sum, difference etc. These primitives are used for deep feature synthesis (DFS), which is the process Featuretools uses to make new features. In my project, the choice of primitives resulted in a total of 3534 total features.

However, before moving forward towards training, the features created from Featuretools needed to be cleaned up, since many features had high missing values or had a high degree of correlation with each other. The low information features (only one unique value) and those that had more than 60% missing values were removed. Then, from the predictor features which had a high degree of correlation with each other, one feature from each feature pair over a set threshold (0.8) were removed. These steps reduced the number of features, which helps reduce the impact of the curse of dimensionality[12]. Thereafter, operations like imputing missing values, one-hot encoding of categorical variables, and rescaling were performed on the remaining features to prepare the dataset for model training.

5.4 Classifier Models: Training, Prediction and Comparison

Before building the full-fledged classification models, I trained a logistic regression model[13] using only the main application dataset, to serve as a low-end benchmark and work as a sanity check. This base case -

logistic regression model, had a high accuracy score of 0.9184. It worked well when predicting the cases when loan is repaid (F1-score = 0.96) but did not perform well to predict when loans are defaulted (F1-score = 0.02), since this is an imbalanced dataset. Hence, I preferred to use AUC ROC which works with data imbalance, since it works on prediction values rather than classes. As shown in Figure 6, the AUC ROC gave a value of 0.7454 for the base case on the test data.

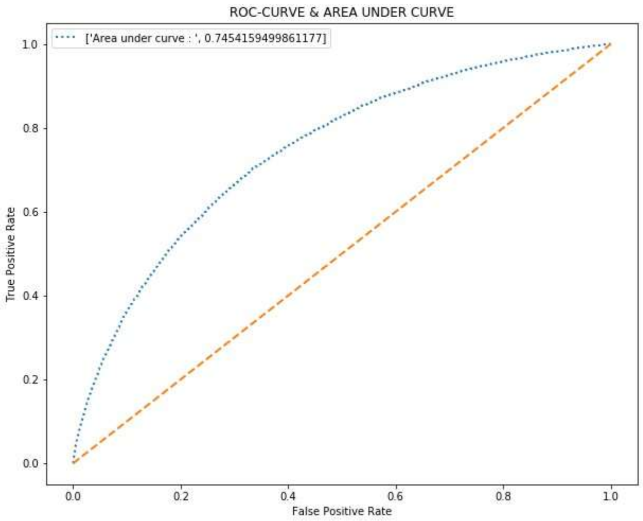


Figure 6- Receiver Operating Characteristic (ROC) and Area Under Curve (AUC) for base case (logistic regression on only training dataset).

The final created dataset is first split into training and testing parts in the ratio 75:25. To reduce the data imbalance where the ratio of the majority class (TARGET=0) to minority class (TARGET=1) is > 10, a technique called random undersampling[5] was implemented on the training dataset. This reduces the skew in data by selecting fewer number of majority class samples, which helps balance the dataset. To prevent losing a large volume of data due to discarding samples from the majority class, a limit was set to consider majority class samples equal to twice the minority class samples. While this does not completely resolve the data imbalance, it makes it more tolerable so that the model is better trained to predict when loans may be defaulted (TARGET=1). For finding the best classifier models, I trained numerous different models on the undersampled training dataset.

5.4.1 Logistic Regression

Logistic regression[13] 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.

$$t = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_n X_n \quad \text{(1A)}$$

$$\sigma(t) = \frac{1}{1+e^{-t}} \quad \text{(1B)}$$

The model from the Scikit-learn[14] package was implemented, and when applied with the default hyperparameters (entire dataset), it gave an accuracy of 0.9184, F1-score of 0.0 and AUC of 0.5079.

5.4.2 Random Forest

Random forest[15] 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.

5.4.3 Decision Tree

Decision tree[16] 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 0.6879, F1-score of 0.1912 and AUC of 0.5806.

5.4.4 Gaussian Naïve Bayes

Gaussian Naïve Bayes[17] 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 0.9184, F1-score of 0.0 and AUC of 0.6691.

5.4.5 XGBoost

XGBoost[8] 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 0.8292, F1-score of 0.3103 and AUC of 0.7604.

5.4.6 Gradient Boosting

Gradient Tree Boosting[18] 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 0.8548, F1-score of 0.3221 and AUC of 0.7735.

5.4.7 LightGBM

LightGBM[9] is a gradient boosting framework that uses tree-based learning algorithm and can be used for supervised classification. It uses histogram-based algorithms, which bucket continuous feature (attribute) values into discrete bins, which decreases training time and memory usage. The model from the LightGBM package was implemented, and when applied with the default hyperparameters and validation split (early stopping), it gave an accuracy of 0.7264, F1-score of 0.2929 and AUC of 0.7806.

The performance of the different classifiers on the test data can be better compared using metrics like accuracy, F1-score, and ROC AUC (Figure 7). We can see from the comparisons of accuracy, F1 score and AUC ROC scores

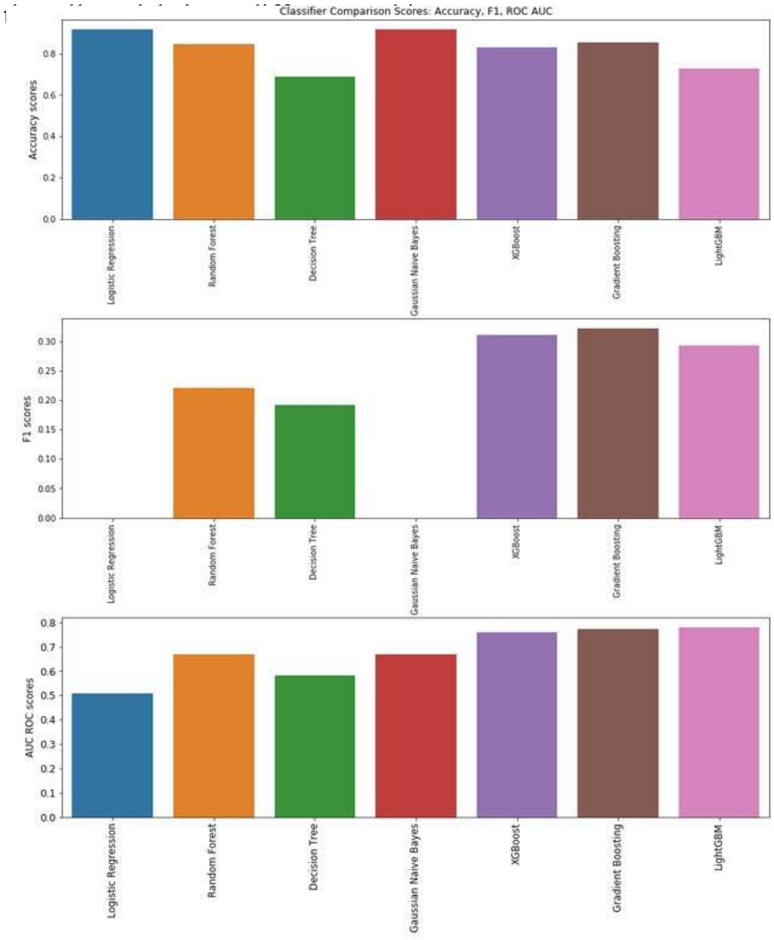


Figure 7- Classifier model comparison metrics: A) Accuracy, B) F1-score, C) ROC AUC score

The Logistic Regression and Gaussian Naive Bayes models have highest accuracy, but the lowest F1 scores. This shows that they do not work well for imbalanced data. The XGBoost, Gradient Boosting and LightGBM classifiers give good F1 and AUC results. The AUC scores can be better visualized using the ROC curves (Figure 8). The AUC scores improve as the curves rise higher above the diagonal. Since, the LightGBM classifier has the best score for AUC (0.781), I selected it moving forward.

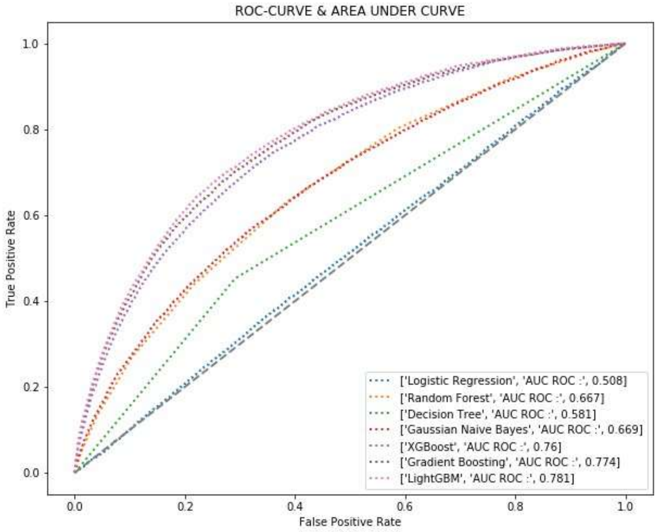


Figure 8- ROC AUC comparison for different classifier models.

After choosing the best classifier, I used K-fold cross validation to select the best model. K-fold cross validation[19] partitions the samples into K subsamples, where in each case one subsample is used for validation while other partitions are used for training. This allowed me to choose parameters corresponding to the best performance (AUC score: 0.7834) without creating a separate validation dataset. At this stage, I decided to compare the top 15 features of the LightGBM model after crossvalidation with that of the base-case model (Figure 9).

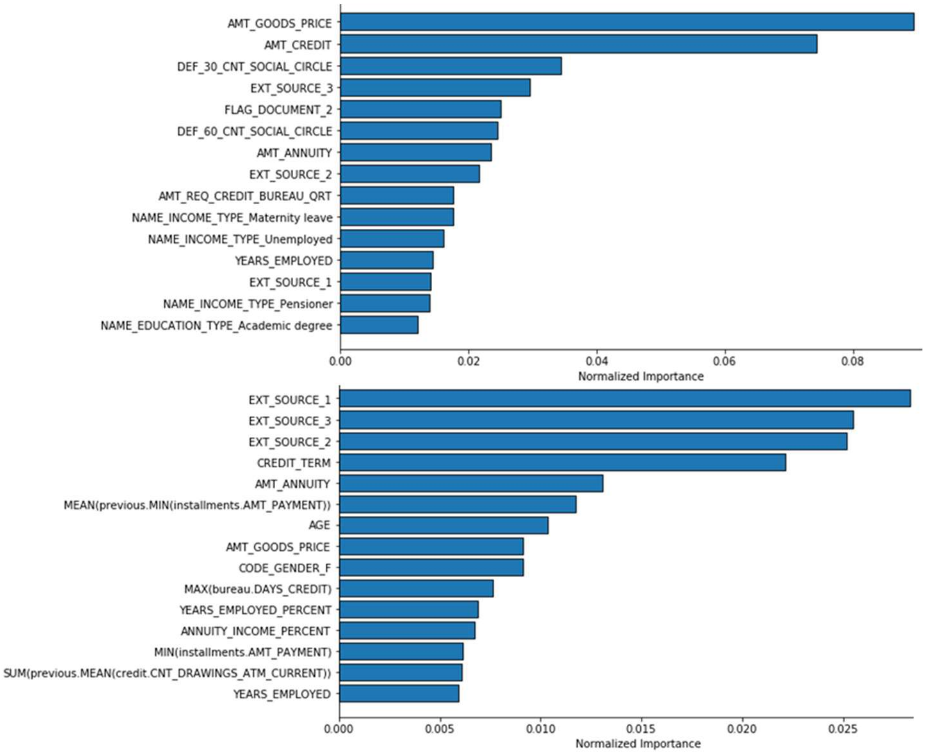


Figure 9- Comparing top 15 features: A) Base case, B) LightGBM model after K-fold cross-validation

From the top 15 features of the base case model (Figure 9A), I observed that it contains a number of features that had a high loan default rate during exploratory data analysis. From the top 15 features of the LightGBM model (Figure 9B), it is seen that it contains a number of features that were created from domain knowledge and through the automated feature engineering (DFS) process. Among the original features, many are those that had a high loan default rate during exploratory data analysis.

5.5 Hyperparameter Tuning

After choosing the binary classifier, we can tune the hyperparameters for improving the model results through grid search, random search, and Bayesian optimization (Hypertopt library)[20]. The hyperparameter tuning process will use an objective function on the given domain space, and an optimization algorithm to give the results. The domain space is the range of hyperparameters over which the tuning process will be carried out. The domain space is the range of hyperparameters over which the tuning process will be carried out.

5.5.1 Grid search

The grid search algorithm iterates over each hyperparameter incrementally, one at a time to explore the entire domain space. However, the exhaustive search can be very time consuming and cannot be completed with finite computing resources. Hence, the maximum evaluations were limited to 20 due to time and computational constraints. The best model from grid search gave an AUC score of 0.7840 on the test data set.

5.5.2 Random search

The random search algorithm selects hyperparameters from the domain space in a random manner. With limited resources, it can act as a preferred strategy due to a less exhaustive search and high possibility of finding good hyperparameter combinations. The maximum evaluations were limited to 50 due to time and computational constraints. The best model from random search gave an AUC score of 0.7860 on the test data set.

5.5.3 Bayesian optimization

The Bayesian method, unlike the previous uninformed methods uses the results of the previous iteration to decide the next hyperparameter combination in the domain space. It uses the Hyperopt library using the Tree Parzen Estimator algorithm[20] to select the next hyperparameter values for evaluation in the objective function. The maximum evaluations were limited to 50 due to time and computational constraints. The best model from Bayesian search gave an AUC score of 0.7836 on the test data set.

The ROC AUC validation scores from all three methods for different iterations can be compared to see the trends (Figure 10). The grid search scores (green) do not differ much since during each iteration, only one hyperparameter is incrementally changed with highest validation AUC of 0.7785.

The random search scores (orange) are scattered with no noticeable trend but have the highest validation ROC AUC score (0.7802) simply by finding a favorable combination. The Bayesian search (blue) can be observed to have progressively better scores with increasing iterations as expected with highest validation AUC of 0.7795, and if the number of iterations were increased could have surpassed the score from random method.

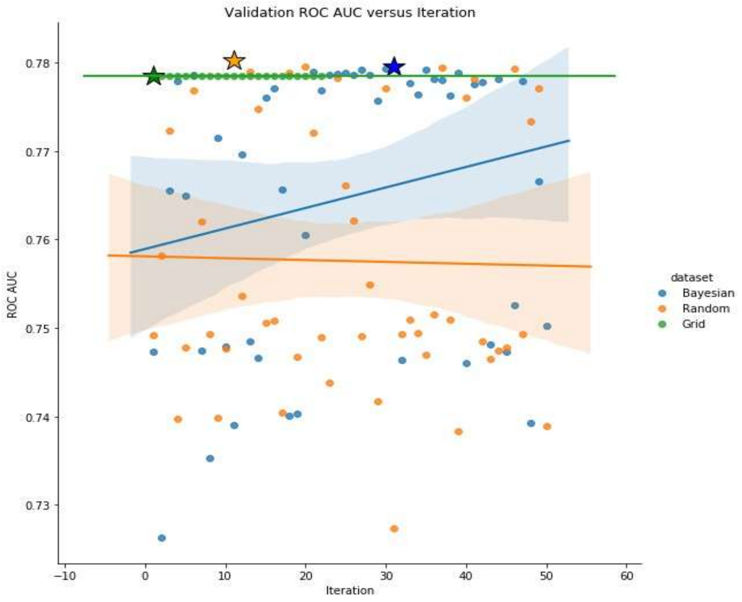


Figure 15- Comparison of validation AUC versus iteration for hyperparameter optimization methods. The star shows the highest AUC for each method.

The hyperparameters for the LightGBM model with best validation AUC (random search) are as shown in Table 2. Finally, the predictions on test dataset from the best model (true/predicted labels and index) were saved.

Hyperparameter	**Value**
boosting_type	gbdt
colsample_bytree	0.6
is_unbalance	False
learning_rate	0.0068724
min_child_samples	445
n_estimators	2897
num_leaves	24
reg_alpha	0.8163265
reg_lambda	0.7346939
subsample	0.6969697
subsample_for_bin	160000

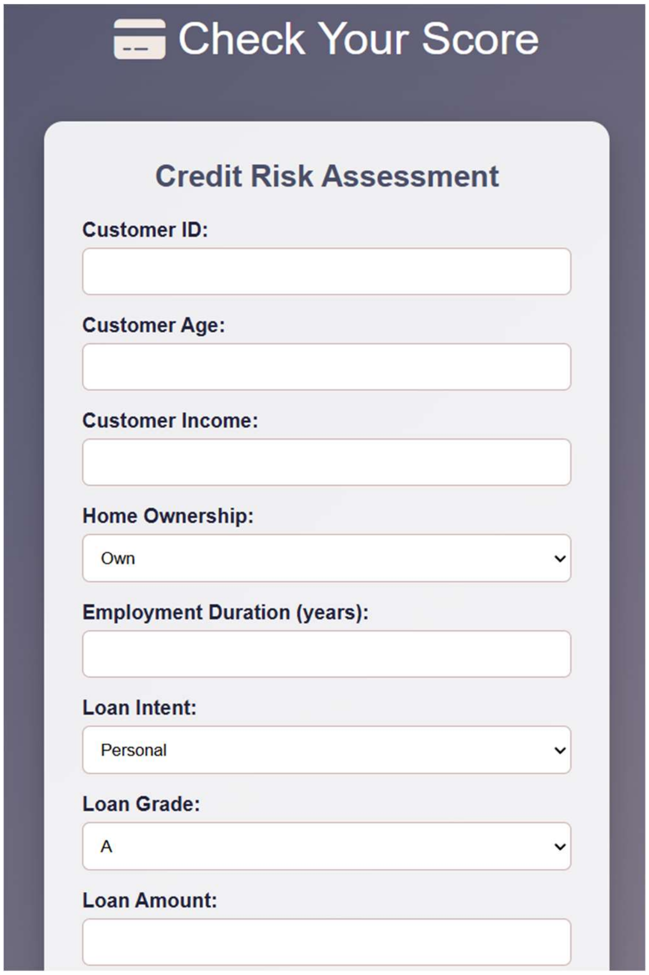
Table 2- Hyperparameters for final prediction model.

6.0 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 leaderboard (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.



Check Your Score

Credit Risk Assessment

Customer ID:

Customer Age:

Customer Income:

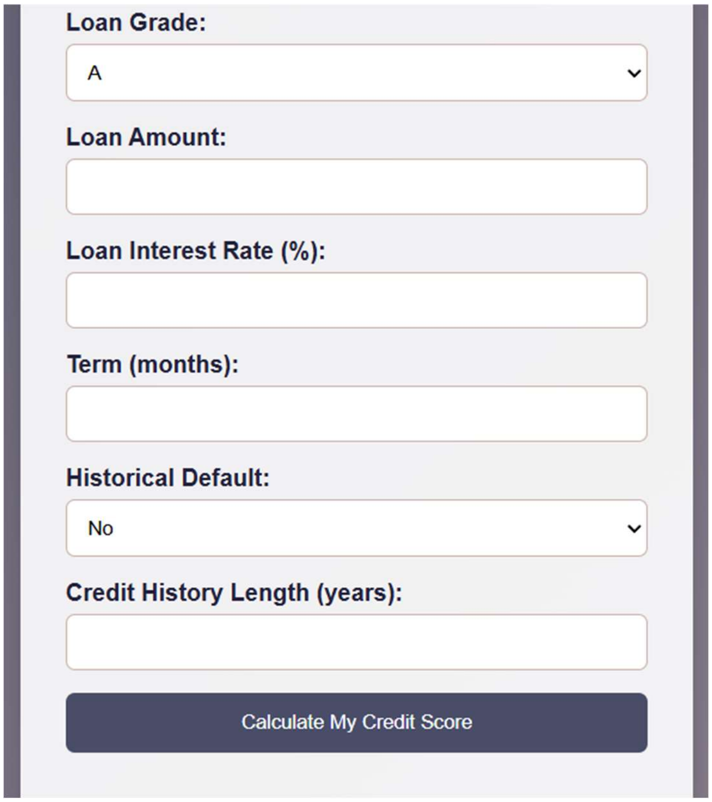
Home Ownership: Own

Employment Duration (years):

Loan Intent: Personal

Loan Grade: A

Loan Amount:



Loan Grade: A

Loan Amount:

Loan Interest Rate (%):

Term (months):

Historical Default: No

Credit History Length (years):

7.0 Acknowledgements

This project builds on scripts and explanations from other Jupyter notebooks publicly shared on Kaggle. The list of notebooks[21-28] is included in the references and in the Jupyter notebook accompanying this report.

8.0 References

1. Home Credit Default Risk Competition (2018). Kaggle. https://www.kaggle.com/c/home-creditdefault-risk/overview
2. Bagherpour, A. (2017). Predicting mortgage loan default with machine learning methods. University of California/Riverside.
3. Khandani, A. E., Kim, A. J., & Lo, A. W. (2010). Consumer credit-risk models via machinelearning algorithms. Journal of Banking & Finance, 34(11), 2767-2787.
4. Ma, X., Sha, J., Wang, D., Yu, Y., Yang, Q., & Niu, X. (2018). Study on a prediction of P2P network loan default based on the machine learning LightGBM and XGboost algorithms according to different high dimensional data cleaning. Electronic Commerce Research and Applications, 31, 24-39.
5. He, H., & Ma, Y. (Eds.). (2013). Imbalanced learning: foundations, algorithms, and applications. John Wiley & Sons
6. Lipton, Z. C., Elkan, C., & Naryanaswamy, B. (2014, September). Optimal thresholding of classifiers to maximize F1 measure. In Joint European Conference on Machine Learning and Knowledge Discovery in Databases (pp. 225-239). Springer, Berlin, Heidelberg.
7. Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B. E., Bussonnier, M., Frederic, J., ... & Ivanov, P. (2016, May). Jupyter Notebooks-a publishing format for reproducible computational workflows. In *ELPUB* (pp. 87-90).
8. Chen, T., & Guestrin, C. (2016, August). XGBoost: A scalable tree boosting system. In Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining (pp. 785-794).
9. Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., ... & Liu, T. Y. (2017). Lightgbm: A highly efficient gradient boosting decision tree. In Advances in neural information processing systems (pp. 3146-3154).
10. Poulos, J., & Valle, R. (2018). Missing data imputation for supervised learning. Applied Artificial Intelligence, 32(2), 186-196.
11. Kanter, J. M., & Veeramachaneni, K. (2015, October). Deep feature synthesis: Towards automating data science endeavors. In 2015 IEEE international conference on data science and advanced analytics (DSAA) (pp. 1-10). IEEE.

12. Bellman, R. (1966). Dynamic programming. Science, 153(3731), 34–37.
13. Wright, R. E. (1995). Logistic regression.
14. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Vanderplas, J. (2011). Scikit-learn: Machine learning in Python. the Journal of machine Learning research, 12, 2825-2830.
15. Liaw, A., & Wiener, M. (2002). Classification and regression by random forest. R news, 2(3), 18-22.
16. Quinlan, J. R. (2014). C4. 5: programs for machine learning. Elsevier.
17. Zhang H. (2004). The optimality of Naive Bayes. Proc. FLAIRS.
18. Ridgeway, G. (2007). Generalized Boosted Models: A guide to the gbm package. Update, 1(1), 2007.
19. Hastie, T., Tibshirani, R., & Friedman, J. (2009). The elements of statistical learning: data mining, inference, and prediction. Springer Science & Business Media.
20. Bergstra, J., Yamins, D., & Cox, D. (2013, February). Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In International conference on machine learning (pp. 115-123).
21. A Gentle Introduction. (2018). Kaggle. https://www.kaggle.com/willkoehrsen/start-here-a-gentleintroduction
22. Introduction to Automated Feature Engineering. (2018). Kaggle. https://www.kaggle.com/willkoehrsen/automated-feature-engineering-basics
23. Advanced Automated Feature Engineering. (2018). Kaggle. https://www.kaggle.com/willkoehrsen/tuning-automated-feature-engineering-exploratory
24. Intro to Model Tuning: Grid and Random Search. (2018). Kaggle. https://www.kaggle.com/willkoehrsen/intro-to-model-tuning-grid-and-random-search
25. Automated Model Tuning. (2018). Kaggle. https://www.kaggle.com/willkoehrsen/automatedmodel-tuning
26. Home Credit Default Risk Extensive EDA. (2018). Kaggle. https://www.kaggle.com/gpreda/home-credit-default-risk-extensive-eda
27. Home Credit Default Risk: Visualization & Analysis. (2018). Kaggle. https://www.kaggle.com/charlievbc/home-credit-default-risk-visualization-analysis
28. Loan repayers v/s Loan defaulters - HOME CREDIT. (2018). Kaggle. https://www.kaggle.com/pavanraj159/loan-repayers-v-s-loan-defaulters-home-credit