Predictive Analysis of Credit Default Risk

Sujan Dutta
sd2516@rit.edu

Amulya Saxena
as7108@rit.edu

I. Introduction

In the evolving financial landscape, managing credit risk is crucial for financial institutions. This project aims to leverage predictive analytics for identifying potential credit defaulters using a real-world, large-scale dataset. Early identification of such risks can aid in implementing proactive measures, thus reducing financial losses. Our contributions include: (1) extracting and engineering relevant features from raw data and (2) using these features to train a machine-learning model to predict loan defaulters. All the Python scripts are published on GitHub [2].

II. Dataset

We consider a large-scale dataset of the customer credit history of a financial institution available on Kaggle [1]. This dataset contains more than 887K instances of loan applications, and each instance contains 74 different factors associated with the loan application and the customer's credit history. These factors represent vital information, including loan amount, loan term, employment, and annual income, which might help decide the risk associated with the loan. The target variable in the dataset is denoted by loan status, which tells if the loan is paid or charged off.

III. Methodology

A. Feature Selection and Engineering

As highlighted in section II, the original dataset features more than 70 columns. During the initial exploration, we realized that many of these columns carry little to no useful information for modeling the loan status. So, we decided to remove the redundant (or irrelevant) columns and processed some columns to extract useful information. The following is a detailed description of the feature selection and engineering process.

- ***Removing sparse features:*** First, we get rid of all the features that are extremely sparse (more than 90% missing data) as they would be incredibly difficult to learn. This step allowed us to reduce 17 features.
- ***Removing redundant features:*** A careful exploration of the data revealed multiple redundant features. We first remove the unique identifiers (*id* and *member id*). We also remove text metadata related to the loan application, including *application URL* and *loan description*. It is worth mentioning that we keep the *loan purpose* feature, which retains the key information from the *loan description*. Then, we remove another text-based feature – *employment title*. This feature is redundant because the borrowers' *annual income* and *employment length* are already present in the dataset. Next, we found some payment-related features interrelated by simple arithmetic operations. For instance, *total payment = total received interest + total received principal + total received late fee*. To avoid redundancy, we remove the following columns – *total received interest, total received principal, outstanding principal*, and *outstanding investor principal*. The dataset contains two geographical features – *state* and *ZIP code*. As these two are highly correlated, we decided to remove the *state* feature to reduce the complexity (and variance) of the model. Finally, we remove features with very low (~ 0) variance, such as *policy code*, *application type*, and *payment plan*.
- ***Feature extraction and engineering:*** To extract useful information from text-based categorical features, we apply label encoding. Following are some label-encoded features – *home ownership, loan purpose, employment verification status, loan grade*, etc. We implement a custom function called `compute_time_since` to transform the date features (e.g., *loan issue date, last payment date, earliest credit line date*, etc.) into more manageable numerical features. More specifically, this function returns the time passed since the input date (from today) in years. Another approach we tried for date columns is the following. We extract the month and year information from the dates to create additional numerical features. We also performed mean imputation on the numerical features (e.g. *annual income, employment length, etc.*) with missing data.
- ***Encoding target variable and preventing data leakage:*** In the original dataset, the target variable, *loan status*, features ten different categories - *Fully Paid, Charged Off, Current, Default, Late, Grace Period, etc.* As we are interested in predicting risky loans, we decided to binarize these categories into two classes, namely, *non-risky* and *risky*. The *non-risky* class entails *Fully Paid and Current* loans, while the *risky* class captures the rest. The *non-risky* class is encoded as 0 and the *risky* class is encoded as 1. We also found some columns that could potentially lead to data leakage. For instance, a non-zero value of *gross recoveries and recovery fee* features would indicate the loan has been charged off. To avoid such data leakage, we removed these columns.

B. Modeling and Experimental Design

After the steps mentioned in section III-A, we are left with 39 features and a binary target. So, the learning task here is a binary classification problem. Here, it is worth noting the significant imbalance in the data with less than 10% of the data belonging to the minority class *risky loans*. In this project, we

implement two different classification algorithms – Random Forest and Neural Network.

- ***Approach 1: Random Forest***

To implement the Random Forest classifier, we utilized Scikit-learn's `RandomForestClassifier` module. We trained the model on 80% of the data and kept the rest for testing. As the data is imbalanced, we weigh the classes accordingly by setting `class_weight` to `"balanced"`. To select the best model, we perform a grid search on the following hyperparameters – *number of trees, splitting criterion,* and *max tree depth*. Table I lists the optimal hyperparameters set found by the grid search.

- ***Approach 2: Neural Network***

Similar to approach 1, we first split the data into training and testing using the train test split function from scikit learn. 80% of the data is used for training (train), and 20% is used for testing (test). The data is stratified based on the *loan status* column to maintain the class distribution in both sets. The data is normalized using `MinMaxScaler`. This ensures that all features are on a similar scale, preventing some features from dominating others during model training. A simple deep neural network (DNN) model is defined using Keras Sequential API. The model architecture consists of three layers: two dense layers with ReLU activation functions and dropout layers for regularization, and a final dense layer with a sigmoid activation function for binary classification. We chose the well-known Adam optimizer and binary cross-entropy loss function to train the network. To handle class imbalance, class weights are calculated. The weights are inversely proportional to the class frequencies, aiming to give more importance to the minority class during training. Class weights are applied during model training using the `class_weight` parameter. The model is trained using the `fit` method of the `KerasClassifier`. Early stopping is implemented to prevent overfitting. Training will stop if the monitored metric (`val_prc` in this case) does not improve for a certain number of epochs (patience) and will restore the best weights.

TABLE I
OPTIMAL HYPERPARAMETERS FOUND USING GRID SEARCH FOR THE RANDOM FOREST MODEL

Hyperparameters	Search Space	Optimal Value
Number of trees	{100, 200, 300, 400, 500}	300
Splitting criterion	{Gini Index, Entropy}	Gini Index
max tree depth	{1, 2, 3, 4, 5, 6, 7}	7

IV. RESULTS AND DISCUSSIONS

Due to the presence of a significant imbalance in the dataset, we need to look at multiple classification metrics to correctly evaluate the proposed approaches. In this paper, we use the following metrics – *accuracy, precision, recall, F1-score,* and *ROC AUC*. Table II and III present the test set performance of the Random Forest and Neural Network respectively. We note that while the Neural Network achieves a higher *ROC AUC* on the test set, the Random Forest demonstrates better *accuracy, precision, recall,* and *F1-score*. We would also like to highlight that both approaches lead to models with high recall and relatively low precision. In other words, the proposed models are very good at detecting nearly all *risky* loans, however, at the cost of a significant number of false positives. So, the financial institutions, if follow these models, would rarely sanction bad loans, however, at the cost of losing some of the valuable customers.

TABLE II
PERFORMANCE EVALUATION OF THE RANDOM FOREST MODEL USING DIFFERENT CLASSIFICATION METRICS

Evaluation metric	Value (%)
Accuracy	91.97
Precision	48.95
Recall	89.42
F1-score	63.27
ROC AUC	90.80

TABLE III
PERFORMANCE EVALUATION OF THE NEURAL NETWORK MODEL USING DIFFERENT CLASSIFICATION METRICS

Evaluation metric	Value (%)
Accuracy	88.15
Precision	20.74
Recall	80.77
F1-score	33.01
ROC AUC	92.25

V. CONCLUSION AND FUTURE WORK

In conclusion, our project demonstrates the use of machine learning models to predict loan defaulters. Through meticulous feature selection and engineering, we were able to create robust models that provide valuable insights into credit risk assessment. We also performed hyperparameter tuning to find the best model. The results across multiple evaluation metrics highlight the potential of predictive analytics in improving financial decision-making and risk management. Future work should explore other advanced machine-learning algorithms, such as – Adaptive Boosting and Extreme Gradient Boosting, to further improve the performance. It would also be interesting to see how the proposed feature engineering methods and models perform in other related datasets.

VI. INDIVIDUAL CONTRIBUTIONS

Contribution	Sujan	Amulya
Feature Engineering	✓	✓
Random Forest Modeling	✓	-
Neural Network Modeling	-	✓

REFERENCES

[1] Dataset: https://www.kaggle.com/datasets/ranadeep/credit-risk-dataset
[2] Codebase: https://github.com/xlumzee/MI-Credit-Risk-Analysis