Final Iteration

Zankhana Mehta (002320268)
Rohan Ojha (002253450)
Prateeksha Devikar (002407443)

November 24, 2024

Abstract

This report provides a comprehensive overview of the project titled ***Automated Loan Application Scorecard for Credit Risk Assessment***. It includes the objectives, methodology, results, challenges faced, and recommendations for future work.

Contents

1 Introduction

1.1 Background

The financial institutions (Banks and NBFCs) traditionally used a combination of manual processes and credit scores to assess the risk involved in giving loans to new customers. These methods are not very reliable as there can be human bias, and lack of scalability when number of applicants are way too large.

In this project, we are building an application scorecard to automate the evaluation process for loan applications. This scorecard uses machine learning algorithms where every new applicant gets a score based on their financial history, demographic information, and other relevant features.

This application scorecard feature will help financial institutions make well-informed decisions for the loan approval process while reducing risks and ensuring fair lending practices.

1.2 Objectives

- Build application scorecard for customers taking loans to minimize the financial risk for the bank by accurately predicting the likelihood of loan defaults.
- Develop a machine learning model that can assess loan applicants based on their data and assign a score that reflects their creditworthiness.
- Utilize the scorecard to predict whether a loan applicant should be approved or rejected based on their assigned score.
- Ensure that the scorecard is interpretable, providing insights into which features most influence the loan approval decision.
- Enable banks to make faster and more consistent loan approval decisions, improving operational efficiency.

2 Methodology

2.1 Introduction to Methodology

In this project, we utilized Python for data preprocessing, feature engineering, and machine learning, and PowerBI for visualization. The objective was to automate the underwriting process using a combination of data-driven insights and predictive modeling. The LendingClub's accepted dataset from Kaggle was used. Python libraries like pandas, numpy, and sklearn streamlined the implementation.

2.2 Data Collection and Preparation

2.2.1 Data Sources

- Used dataset from Kaggle `https://www.kaggle.com/datasets/wordsforthewise/lending-club?select=accepted_2007_to_2018Q4.csv.gz`
- Used their accepted dataset consisting of 151 columns.

2.2.2 Data Transformation

- Created additional column 'Bad_Flag' to identify defaulters.
- This column has binary values where 1 indicates customers who charged off and 0 indicates the rest.

2.2.3 Dropping variables

- Initially, after going through all the variables, we decided to drop certain variables.
- These variables are insignificant in predicting target values.

2.2.4 Missing Value Analysis and Imputation

- By doing missing analyzer, we could see how much percent missing values each variable has.
- Categorized variables according to their missing value percentages: Category 1 - 0% > and < 50%, Category 2 - 50% >= and < 80%, Category 3 - > 80%.
- We decided to drop variables belonging to Category 3.
- For Category 2, since they have delinquency columns, we imputed the values as NULL.
- For Category 1 variables, we created bins and imputed median value of the non-null bucket, where: Bad Rate of Non-Null bucket (is almost) = Bad Rate of Null bucket.

2.2.5 Outlier Treatment

- We performed Outlier treatment on all remaining numerical variables.

2.3 Feature Engineering

2.3.1 Distribution of Categorical variables

- We created distribution plots for all categorical variables.
- After going through the distribution plots, we compared it with our intuition and decided to keep 5 variables.
- These variables will go through One-Hot Encoding.

2.3.2 One-Hot Encoding

- Performed one-hot encoding for selected 5 variables.

2.3.3 Quick and Dirty Model for variable selection

- We used quick and dirty model to understand the relationship between predictors and target.
- This helps us focus on variables with high predictive power for the target
- Evaluation metrics used are KS and Gini coefficients.
- Higher KS and Gini value variables have more predictive power.
- This step is done to get a clear understanding of feature relevance in a short timeframe.

2.3.4 Correlation analysis and multicollinearity

- Selected some variables based on KS, GINI, and Correlation values.
- Finalized variables that made sense in the final modeling process.
- Finally, we checked VIF among predictors through an iterative process.

2.4 Algorithm Development

2.4.1 Model Selection

- We implemented three machine learning models: Ridge Regression, Random Forest, GBM
- Ridge Regression gave a Test Accuracy of 0.9662 and Loss of 0.0822.
- Random Forest gave a Test Accuracy of 0.9841 and Loss of 0.0516.
- GBM gave a Test Accuracy of 0.9753 and Loss of 0.0731.
- We also calculated the accuracy of Random Forest with fewer predictors. It gave accuracy of 0.9829 and loss of 0.0627.

- The predictive power of the model remains unchanged. This makes the model more explainable to the business/stake-holders.
- Thus, Random Forest was chosen.

2.4.2 OOT Analysis

- We performed OOT on data from Jan 2015 to Mar 2015.
- Ridge Regression gave a Test Accuracy of 0.9641 and Loss of 0.0855.
- Random Forest gave a Test Accuracy of 0.9836 and Loss of 0.0618.
- GBM gave a Test Accuracy of 0.9741 and Loss of 0.0759.

2.5 Data Visualization and Insights

2.5.1 Visual Analytics

- We used Python libraries like matplotlib and seaborn.
- These libraries were used to study distribution and trend analysis of variables.

2.5.2 Enhanced Understanding through visualization

- We used PowerBI to make interactive dashboards.

3 Scorecard Generation

- We have selected Random Forest model since it came out to be the most accurate one among 3 models.
- Scorecard is built based on the probabilites obtained from the model.
- Greater the score, lesser are the chances of defaulting.
- Minimum and maximum score came out to be 300 and 850 respectively.

4 Results

- The graph compares the training errors and OOT errors for the three models: Ridge Regression, Random Forest, and Gradient Boosting Machine (GBM).
- Ridge Regression gives the highest training and OOT error. The gap between training and OOT errors is minimal, that means it is underfitting and the model is not capturing complex data patterns effectively.

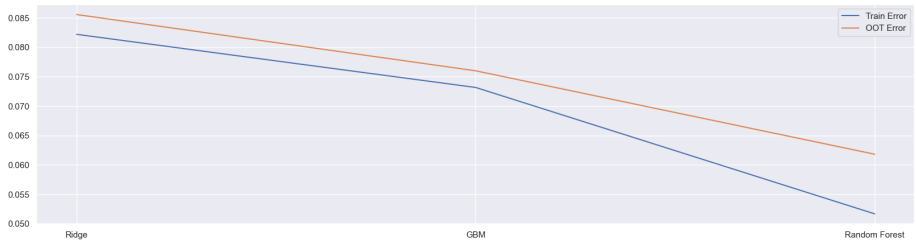

Figure 1: Training error and OOT error for three models

- GBM has a lower gap between the training and OOT errors indicating good generalization.
- Random Forest shows lowest OOT error of all models, making it the best choice in terms of out-of-sample performance. The training error decreases steadily, and the gap between training and OOT errors is relatively small, indicating that the model is well-regularized and not overfitting.

5 Challenges and Limitations

Describe the difficulties encountered and their impact on the project.

6 Conclusion

Random Forest exhibited the lowest OOT error, demonstrating the best generalization and predictive power among the models tested. Its relatively small gap between train error and OOT error indicates that it balances complexity and regularization well, making it the most suitable for deployment.

7 Project Repository

The project code and resources are available on GitHub: `https://github.com/zankhana46/ApplicationScorecard/tree/main/Iteration3`

References

- Numpy `https://numpy.org/doc/`
- Pandas `https://pandas.pydata.org/docs/`
- Matplotlib `https://matplotlib.org/stable/`
- Seaborn `https://seaborn.pydata.org/`

- Scikit-learn https://scikit-learn.org/stable/
- PowerBI https://docs.microsoft.com/en-us/power-bi/
- Ridge Regression https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html
- Random Forest https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
- GBM https://scikit-learn.org/dev/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html