Lend or Lose - ML Project

Ayush Arya Kashyap (IMT2022129)
Uttam Hamsaraj (IMT2022524)
Keshav Goyal (IMT2022560)

Contents

1. Introduction

Financial loan services play a critical role in industries ranging from banking to government finance, aiming to minimize payment defaults and ensure timely loan repayments. To achieve this systematically and efficiently, companies increasingly rely on machine learning techniques to predict individuals at high risk of defaulting. These insights enable targeted interventions, reducing losses and improving financial outcomes.
The dataset used in this project consists of 255,347 entries and 18 features, providing a unique opportunity to address a practical machine learning problem. By leveraging this dataset, we aim to develop and evaluate models that can accurately predict loan default risks, contributing to more robust decision-making processes in the financial sector.

2. Preliminary Steps

This project aims to predict loan defaults using a dataset containing financial and personal details of loan applicants. To address this, we applied various machine learning models after data preprocessing and resampling techniques. Key libraries used :

- `pandas`, `numpy`: For data manipulation and analysis.
- `seaborn`, `matplotlib`: For data visualization.
- `scikit-learn`: For machine learning model implementation and evaluation.
- `imbalanced-learn` (`SMOTE`): For handling class imbalance in the dataset.

3. Preprocessing

The preprocessing phase involved several crucial steps to prepare the dataset for modeling:

1. **Handling Missing and Duplicate Values:** We examined the dataset for missing values and duplicate entries. As none were found, no further action was necessary in this regard.
2. **Correlation Analysis:** We computed the correlation between features to identify relationships and determine feature importance. One of the columns, *LoanTerm*, was removed due to its low correlation with the target variable, as it did not contribute significantly to predictive performance.
3. **Data Visualization for Outliers:** Outliers were identified through visual analysis using boxplots and scatter plots. These insights helped ensure data consistency and quality. As none were found in this case as well, no further action was required in that regard as well.
4. **Normalization:** To standardize the data and improve model performance, numerical features were normalized to ensure they were on a comparable scale. We tried implementing Scaling instead of Normalization as well but Normalization was the one which gave better results.
5. **Encoding:** For both multi-class and binary-class categorical features, we used Label Encoding to convert each category into a numerical feature. This was necessary because those columns contained strings, which needed to be converted to integers so the model could handle them effectively.

Below are some visualizations illustrating the preprocessing steps:

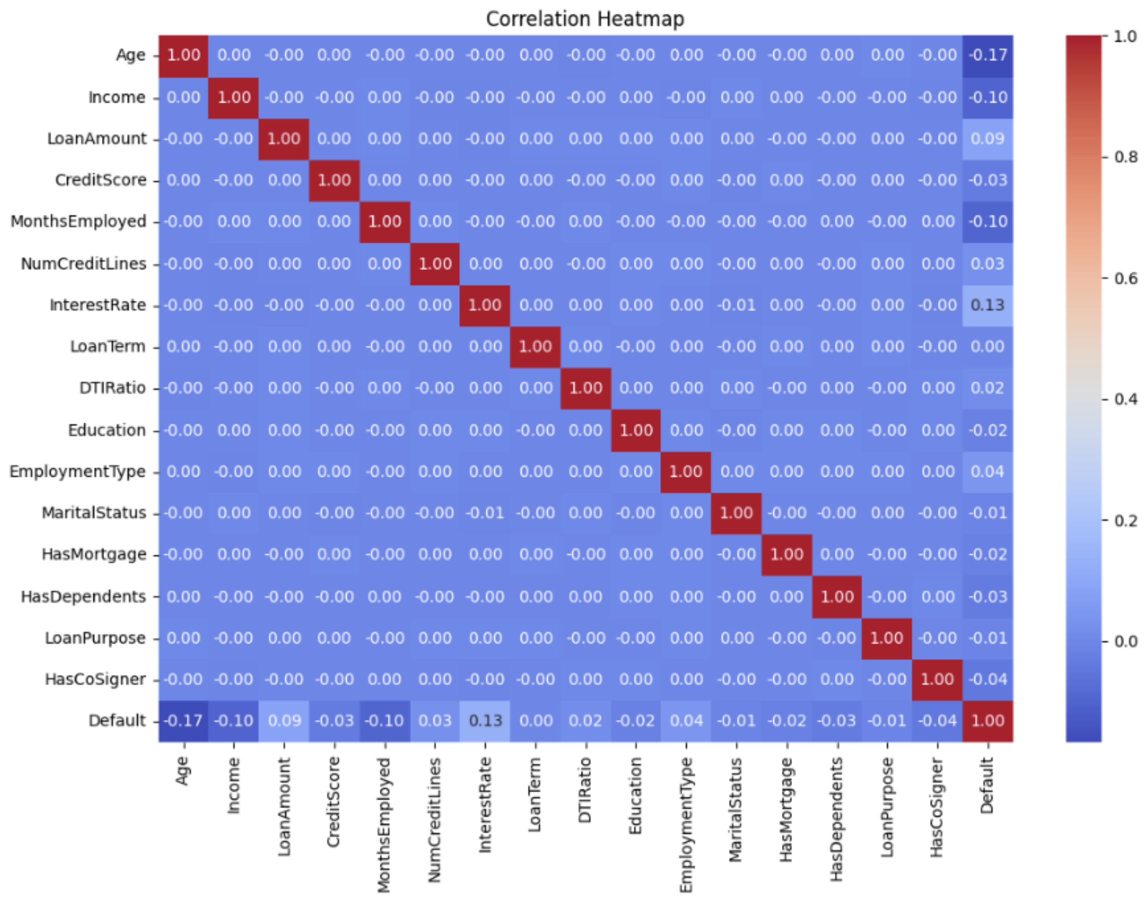
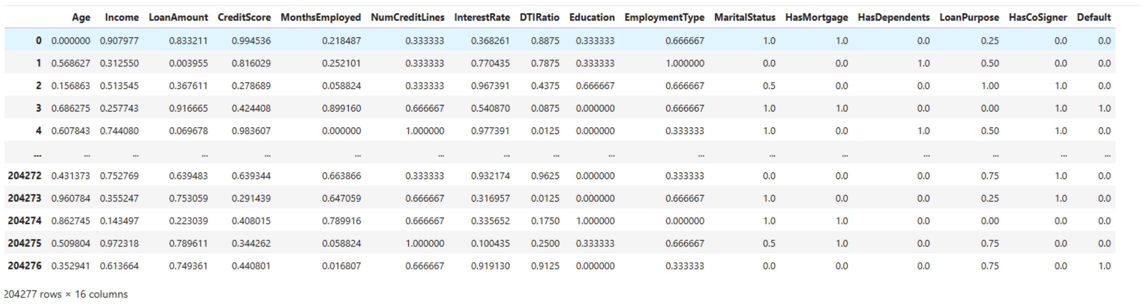


Figure 1. Correlation Matrix of Features

Figure 3a shows how LoanTerm has less correlation and thus can be removed.



	Age	Income	LoanAmount	CreditScore	MonthsEmployed	NumCreditLines	InterestRate	DTIRatio	Education	EmploymentType	MaritalStatus	HasMortgage	HasDependents	LoanPurpose	HasCoSigner	Default
0	0.000000	0.907977	0.833211	0.994536	0.218487	0.333333	0.368261	0.8875	0.333333	0.666667	1.0	1.0	0.0	0.25	0.0	0.0
1	0.568627	0.312550	0.003955	0.816029	0.252101	0.333333	0.770435	0.7875	0.333333	1.000000	0.0	0.0	1.0	0.50	0.0	0.0
2	0.156863	0.513545	0.367611	0.278689	0.058824	0.333333	0.967391	0.4375	0.666667	0.666667	0.5	0.0	0.0	1.00	1.0	0.0
3	0.686275	0.257743	0.916665	0.424408	0.899160	0.666667	0.540870	0.0875	0.000000	0.666667	1.0	1.0	0.0	0.00	1.0	1.0
4	0.607843	0.744080	0.069678	0.983607	0.000000	1.000000	0.977391	0.0125	0.000000	0.333333	1.0	0.0	1.0	0.50	1.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
204272	0.431373	0.752769	0.639483	0.639344	0.663866	0.333333	0.932174	0.9625	0.000000	0.333333	0.0	0.0	0.0	0.75	1.0	0.0
204273	0.960784	0.355247	0.753059	0.291439	0.647059	0.666667	0.316957	0.0125	0.000000	0.666667	1.0	0.0	0.0	0.25	1.0	0.0
204274	0.862745	0.143497	0.223039	0.408015	0.789916	0.666667	0.335652	0.1750	1.000000	0.000000	1.0	1.0	0.0	0.00	0.0	0.0
204275	0.509804	0.972318	0.789611	0.344262	0.058824	1.000000	0.100435	0.2500	0.333333	0.666667	0.5	1.0	0.0	0.75	0.0	0.0
204276	0.352941	0.613664	0.749361	0.440801	0.016807	0.666667	0.919130	0.9125	0.000000	0.333333	0.0	0.0	0.0	0.75	0.0	1.0

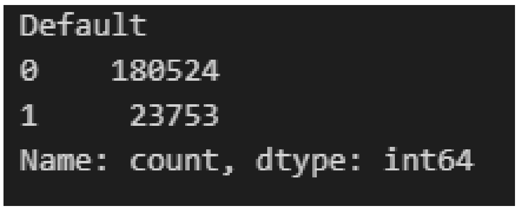
204277 rows × 16 columns

Figure 2. Effect of Normalization on Numerical Features

Through these preprocessing steps, we ensured that the dataset was clean, consistent, and ready for model training and evaluation.

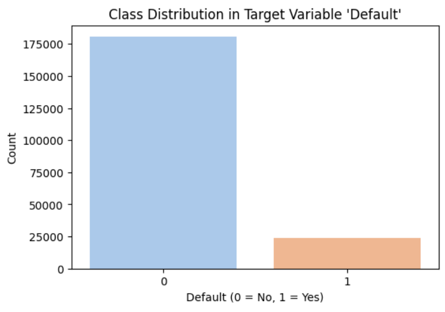
4. Resampling (SMOTE)

The dataset was highly imbalanced with respect to the target variable (`Default`). Below shown images support our conclusion:



```
Default
0    180524
1     23753
Name: count, dtype: int64
```

(a) Number of each class in Default



(b) Data Distribution Visualization

Figure 3. Image showing the imbalance in distribution in the Default column.

To tackle this, we could've used one of the following techniques:

- **Undersampling:** This method reduces the number of samples from the majority class to balance the class distribution. While it can lead to loss of valuable data, it helps make the dataset more manageable.
- **Oversampling:** This technique increases the number of samples in the minority class by duplicating existing data points. It can help balance the dataset but may cause overfitting due to repeated data.
- **SMOTE (Synthetic Minority Over-sampling Technique):** SMOTE creates synthetic samples by interpolating between existing minority class samples. This method not only balances the classes but also helps the model generalize better by generating more diverse examples.

We ended up using **SMOTE** since such a large dataset required a lot of computation time to apply Undersampling or Oversampling.

5. Models Tried

Several machine learning models were evaluated to predict loan defaults:

- **Decision Tree Classifier**: Simple, interpretable, and performed well as a baseline model. Each decision node splits the data based on the most significant feature.
- **Random Forest Classifier**: An ensemble approach to improve performance with multiple decision trees. It reduces overfitting by averaging the results of many diverse trees.
- **Extra Trees Classifier**: A variant of Random Forest, where splits are made extremely random, making the model faster and often providing similar or better accuracy. It decreases variance by averaging predictions from a set of highly randomized trees.
- **Gradient Boosting Classifier**: Applied for higher accuracy, this model iteratively improves by learning from previous errors. It combines weak learners to form a more accurate model.
- **AdaBoost Classifier**: A boosting algorithm that combines weak learners (usually decision trees) to form a strong classifier, focusing on the errors of previous models to improve accuracy. Each subsequent model is trained to correct errors made by the previous ensemble.

- **LightGBM Classifier**: A gradient boosting framework that uses tree-based learning algorithms, optimized for speed and efficiency on large datasets, with reduced memory usage. It employs a leaf-wise growth strategy, allowing it to converge faster than traditional boosting methods.
- **XGBoost Classifier**: An advanced implementation of gradient boosting designed for performance and speed. It uses regularization techniques to reduce overfitting and handles missing data automatically, making it a highly efficient choice for large datasets with complex relationships.
- **Polynomial Regression**: An extension of linear regression that models the relationship between the independent variable(s) and the dependent variable as a n-degree polynomial. It is useful for capturing non-linear relationships by fitting a curve to the data. The model works by adding polynomial terms of the input features, allowing it to fit more complex patterns. We found the best result for n=2.

The accuracy of each model was evaluated, with a focus on maximizing performance while minimizing the risk of overfitting. For hyperparameter tuning, we initially utilized GridSearchCV, a comprehensive search method, to find the optimal parameters. However, due to the substantial computational time required—exceeding 8 hours for models like Random Forest and Gradient Boost—we limited the use of GridSearchCV to simpler models like Decision Trees. For more complex models, we opted for manual hyperparameter tuning, adjusting parameters iteratively to achieve the best balance of accuracy and efficiency.

6. Kaggle Submission Stats

Model	**Original**		**PCA**	
	Test Accuracy (%)	**Kaggle Score**	**Test Accuracy (%)**	**Kaggle Score**
XGBoost	88.50	88.801	88.29	88.447
LightGBM	88.52	88.760	88.29	88.447
Polynomial Regression	88.67	88.748	88.29	88.529
Random Forest	88.49	88.723	87.90	88.049
GradientBoost	88.48	88.720	88.28	88.320
Extra Trees	88.36	88.507	87.77	88.011
Decision Tree	88.08	88.259	88.10	88.29
AdaBoost	80.00	80.290	78.97	79.500

Table 1. Comparison of model performance with and without PCA

7. Best Submission

After evaluation, Gradient Boosting Classifier emerged as the best model in terms of accuracy and generalization. The model, combined with SMOTE resampling and optimized hyperparameters, showed the highest precision and recall on the validation set. We ended up with a score of *88.801* on kaggle, *0.028* behind the leaders who had an accuracy of *88.829*.

8. Analyzing the Final Model

XGBoost consistently outperformed other models, achieving the highest accuracy scores. The model's ability to handle complex relationships between features and its built-in regularization made it the best choice for our loan prediction task.

9. Important Links and Documents

- Dataset Source: https://www.kaggle.com/competitions/lend-or-lose/data
- Project Code Repository: https://github.com/ayusharyakashyap/Loan-Default-Prediction