ML Project Report: Lend or Lose

Team Members

1. Trupti Khodwe - IMT2022007
2. Sourav M Dileep - IMT2022033
3. Rishabh Dixit - IMT2022051

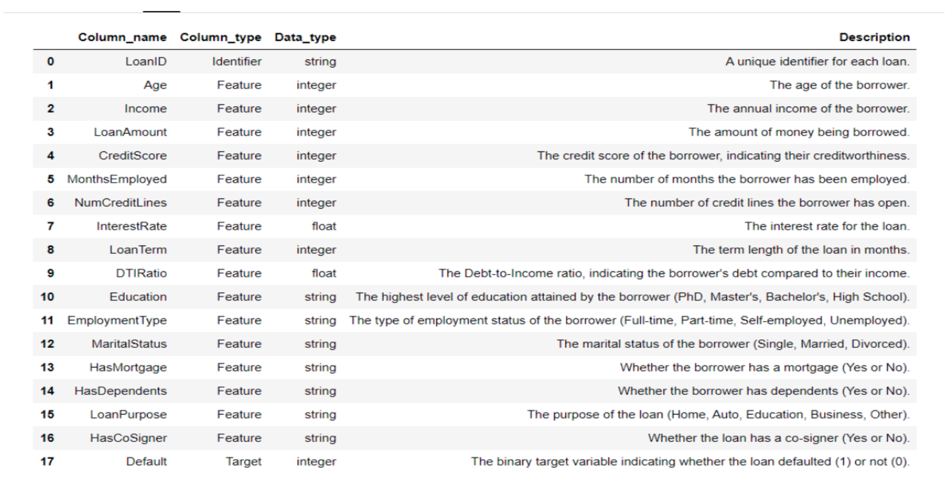
Github Link: https://github.com/truptikhodwe/Lend-or-Lose

Task

Training ML models to predict which individuals are at the highest risk of defaulting on their loans is essentially a **binary classification problem**, where the target variable is the likelihood of default. Several factors, as mentioned below, contribute to predicting whether an individual is likely to repay a loan or not, making this task a matter of differentiating between two possible outcomes: default (1) or no default (0).

Dataset and Features Description

Financial loan services are leveraged by companies across many industries, from big banks to financial institutions to government loans. One of the primary objectives of companies with financial loan services is to decrease payment defaults and ensure that individuals are paying back their loans as expected. In order to do this efficiently and systematically, many companies employ machine learning to predict which individuals are at the highest risk of defaulting on their loans, so that proper interventions can be effectively deployed to the right audience. This dataset has been taken from Coursera's Loan Default Prediction Challenge and will provide you the opportunity to tackle one of the most industry-relevant machine learning problems with a unique dataset that will put your modelling skills to the test. The dataset contains 255,347 rows and 18 columns in total.



	Column_name	Column_type	Data_type	Description
0	LoanID	Identifier	string	A unique identifier for each loan.
1	Age	Feature	integer	The age of the borrower.
2	Income	Feature	integer	The annual income of the borrower.
3	LoanAmount	Feature	integer	The amount of money being borrowed.
4	CreditScore	Feature	integer	The credit score of the borrower, indicating their creditworthiness.
5	MonthsEmployed	Feature	integer	The number of months the borrower has been employed.
6	NumCreditLines	Feature	integer	The number of credit lines the borrower has open.
7	InterestRate	Feature	float	The interest rate for the loan.
8	LoanTerm	Feature	integer	The term length of the loan in months.
9	DTIRatio	Feature	float	The Debt-to-Income ratio, indicating the borrower's debt compared to their income.
10	Education	Feature	string	The highest level of education attained by the borrower (PhD, Master's, Bachelor's, High School).
11	EmploymentType	Feature	string	The type of employment status of the borrower (Full-time, Part-time, Self-employed, Unemployed).
12	MaritalStatus	Feature	string	The marital status of the borrower (Single, Married, Divorced).
13	HasMortgage	Feature	string	Whether the borrower has a mortgage (Yes or No).
14	HasDependents	Feature	string	Whether the borrower has dependents (Yes or No).
15	LoanPurpose	Feature	string	The purpose of the loan (Home, Auto, Education, Business, Other).
16	HasCoSigner	Feature	string	Whether the loan has a co-signer (Yes or No).
17	Default	Target	integer	The binary target variable indicating whether the loan defaulted (1) or not (0).

EDA and pre-processing

The following section outlines the necessary steps for conducting essential Exploratory Data Analysis (EDA) to gain insights, detect potential issues, and prepare data for model training.

1. Import Libraries and Load Dataset

The initial step in the EDA process involves importing essential libraries such as pandas, numpy, matplotlib, seaborn, and relevant modules from scikit-learn for data preprocessing. The dataset is loaded into a pandas DataFrame for further analysis and manipulation.

2. Data Overview

The dataset is examined by displaying the first few rows, checking data types, and identifying any missing values. This step helps in understanding the structure and initial quality of the data. A statistical summary of numerical columns is also obtained to review the central tendency, spread, and potential outliers.

3. Handling Missing Values

To handle missing values, the dataset was thoroughly examined for any null values, confirmed by visualising missing data patterns. A missing data matrix plot was generated, which clearly indicated that all columns were complete, with no missing values present. As a result, no imputation or deletion of columns was necessary. This comprehensive check ensures that the dataset is complete and ready for further preprocessing steps, without any concerns about missing data potentially impacting the model's performance.

4. Handling Duplicate Values

We handled the duplicates by dropping them from the dataset if there were any. We used the drop_duplicates method of the DataFrame object in the pandas library for this purpose.

5. Exploratory Data Analysis (EDA)

Various plots were generated to analyse the data distribution, detect outliers, and explore relationships among features. These visualisations provide critical insights that inform subsequent preprocessing and modelling decisions. Key visualisations and their interpretations are detailed below:

- **Histograms for each feature:**

  For each numerical feature, histograms and kernel density estimates (KDEs) were plotted to observe the data's spread, skewness, and distribution shape.

  Here are some example conclusions we can draw from these histograms:

  ➔ **Age Distribution:**

  The age distribution appears relatively uniform, meaning there's a similar number of individuals across different age groups.

  There are slightly fewer individuals in the younger and older age groups, as shown by the slight dip in the bars at the extremes (around ages 20 and 70).

  Overall, the data indicates a fairly balanced distribution of ages, with no significant peaks or dips across the middle age range.

  ➔ **Income Distribution:**

  The income distribution is also relatively uniform, suggesting that there is a broad and even distribution of income levels in the dataset.

  There are minor fluctuations, with slightly fewer individuals at the very low and very high income ends (around 20,000 and 150,000+), as indicated by the slight dip at the extremes.

  The uniform distribution in income may imply that the dataset is representative of various income levels without any significant clustering in any specific range.

  Both distributions are close to uniform, suggesting that the dataset may have been constructed or sampled to cover a wide range of ages and incomes evenly. This might indicate that the data was generated with a balanced approach across age and income categories.

- **Boxplots for Outlier detection:**

  Boxplots were used to examine each numerical feature for potential outliers, which appear as points beyond the plot's whiskers.

  In this dataset, no significant outliers were observed, as no data points appeared beyond the whiskers of any feature. This indicates that the data is relatively clean and does not have extreme values that could disproportionately influence certain models.

- **Correlation Matrix (Heatmap):**

  The correlation matrix heatmap reveals the relationships between different variables in the dataset. Here are a few key takeaways:

  ➔ **Age and Default:**

  Age has a moderate negative correlation with Default (-0.17), indicating that older individuals are slightly less likely to default on loans.

  ➔ **Income and Default:**

  Income also has a weak negative correlation with Default (-0.1), suggesting that individuals with higher incomes are marginally less likely to default.

  ➔ **Loan Amount and Default:**

  LoanAmount shows a weak positive correlation (0.13). This may indicate that larger loans tend to come with slightly higher interest rates.

  ➔ **Months Employed and Default:**

  MonthsEmployed has a weak negative correlation with Default (-0.095), suggesting that individuals with more stable employment (more months employed) are somewhat less likely to default.

  Overall Conclusion:

  None of the variables have strong correlations with Default, indicating that no single factor in this dataset strongly predicts default on its own. However, factors such as Age, Income,CreditScore and MonthsEmployed show slight negative correlations with Default, suggesting they might be weak predictors of credit risk.

6. Data Preprocessing

In this stage, the data is prepared for modelling through the following steps:

- Encoding categorical variables using label encoding, which converts categorical values to a numerical format suitable for models. We did this because it makes it possible for algorithms to interpret and comprehend the data by giving each category within a variable a distinct number.

- Standardising numerical features using a StandardScaler to ensure all features have a similar range and reduce model sensitivity to feature scale.
- Performing feature engineering if needed, such as creating new features to capture nonlinear relationships between variables.

7. Splitting the Dataset

The dataset is split into training and testing sets, ensuring that model performance can be evaluated on unseen data. A common split ratio is 80% for training and 20% for testing. This step prepares the dataset for model training and evaluation.

Summary

The EDA and preprocessing steps include:

- Inspecting data for types, missing values, and basic statistics.
- Imputing missing values based on column types.
- Visualising distributions, detecting outliers, and understanding feature relationships.
- Encoding and scaling features for model-readiness.
- Splitting the dataset for training and testing, completing the data preparation process.

Find all the preprocessing done here: EDA_and_Preprocessing github

Models Used For Training

1. **Decision Tree**
   a. Preprocessing was the same as how we did in the preprocessing section and for the decision tree model, we trained the model two times, one with feature selection and one without feature selection.
   b. We used GridSearchCV for finding the best hyper parameters in both the models. The best parameters we got are as follows:

      i. **With feature selection:** ‘criterion’: ‘entropy’, ‘max_depth’: 10, ‘min_samples_leaf’: 4, ‘min_samples_split’: 10

      ii. **Without feature selection:** ‘criterion’: ‘entropy’, ‘max_depth’: 10, ‘min_samples_leaf’: 4, ‘min_samples_split’: 10

   c. The accuracy on the validation data when we did not do feature selection was 88.40 and when we did, it was 88.54.

d. We used Recursive feature elimination with cross-validation to select the most important feature. This was the output for the selected feature:
    i. Selected features: Index(['LoanAmount'], dtype='object')
e. We saved the trained model in the finalised_model and we are using that model while testing, and then the results are saved in the submission.csv files.
f. The accuracy we got for the decision tree without feature selection was 87.879 and the one we got for the decision tree with feature selection was 88.388.

2. **K nearest neighbours**
    a. For the KNN model, we followed the same preprocessing steps as outlined in the preprocessing section. Along with this, we trained the model after applying dimensionality reduction through Principal Component Analysis (PCA).
    b. We used GridSearchCV to optimise the hyperparameters for the KNN model. The hyperparameters considered were the number of neighbours (n_neighbors), the distance metric (metric), and the weight function (weights). The best parameters obtained were as follows:
        i. Best Parameters: {'metric': 'manhattan', 'n_neighbors': 11, 'weights': 'uniform'}
    c. The accuracy we obtained for the validation data: 88.44.
    d. We saved the trained model in the finalised_model and we are using that model while testing, and then the results are saved in the submission.csv files.
    e. The accuracy we got was 88.214
    f. Feature Reduction with PCA: We applied PCA to retain 95% of the variance, which resulted in 16 principal components. The dimensionality reduction helped in making the model more efficient while retaining the majority of the information from the features.
    g. Standard Scaler:

       i. fit_transform on the training data calculates the mean and standard deviation for each feature, then scales them. Transform on the test data uses the same mean and standard deviation from the training data to ensure consistency.

3. **Random Forest Classifier**
    a. Preprocessing was the same as how we did in the preprocessing section.
    b. We used GridSearchCV for finding the best hyper parameters. The best parameters we got are as follows:

        i. Best parameters: {'max_depth': 20, 'min_samples_leaf': 2, 'min_samples_split': 10, 'n_estimators': 100}

   c. The accuracy on the validation data was 88.81.
   d. The accuracy we got on the submission.csv file on kaggle was 88.686.

4. **Naive Bayes Classifier**

   Pre-processing is similar to other models (checking for null values, outlier detection etc.). We tested the model with and without feature selection.

   a. For feature selection, we used SelectKBest from and got the following output:
      - Selected Features: Index(['Age', 'Income', 'LoanAmount', 'CreditScore', 'MonthsEmployed', 'NumCreditLines', 'InterestRate', 'DTIRatio', 'Education', 'EmploymentType', 'HasMortgage', 'HasDependents', 'HasCoSigner'], dtype='object')
      - Accuracy: 0.8850597219502644

5. **XGBoost**
   a. The pre-processing involves checking for null values, removing duplicates and removing outlier points.
   b. We used GridSearchCV for hyperparameter tuning and got the following result:
      i. Best parameters: {'colsample_bytree': 0.8, 'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 200, 'subsample': 0.8}
      ii. Best score: 0.8859693655063055
      iii. Test Accuracy of best model: 0.887091247307617
   c. For feature selection we used feature importance plots. By observing the plots for both importance types (‘weight’ & ‘gain’), we realised that ‘LoanTerm’ has the least significance. So we re-trained the model after dropping this feature. The result after applying GridSearchCV is as follows:
      i. Best parameters: {'colsample_bytree': 0.8, 'learning_rate': 0.2, 'max_depth': 3, 'n_estimators': 100, 'subsample': 0.8}
      ii. Best score: 0.8859204140616289
      iii. Test Accuracy of best model: 0.8874339142353632
   d. As we can see the accuracy improved and when we tested this with test.csv, we got the highest accuracy of **0.88791.**

6. **GradientBoosting**

   a. Similar to XGBoost, after preprocessing we used GridSearchCV to find the best parameters for Gradient Boosting.

Best Parameters: {'learning_rate': 0.1, 'max_depth': 3, 'min_samples_split': 5, 'n_estimators': 100, 'subsample': 0.8}
Best Score: 0.8857184813341556
Test Accuracy: 0.8869688662619933

b. For feature selection, we again used feature importance graphs. First we dropped 'LoanTerm' feature, which improved the accuracy slightly:

   Best Parameters: {'learning_rate': 0.1, 'max_depth': 3, 'min_samples_split': 2, 'n_estimators': 100, 'subsample': 0.8}
   Best Score: 0.8857551966962687
   Test Accuracy: 0.8870422948893676

c. We tried to drop more features ('LoanPurpose', 'HasMortgage', 'DTIRatio') to improve the accuracy and got the following result:

   Best Parameters: {'learning_rate': 0.1, 'max_depth': 3, 'min_samples_split': 5, 'n_estimators': 100, 'subsample': 0.8}
   Best Score: 0.8856817658222659
   Test Accuracy: 0.8871401997258664

7. **AdaBoost**
   a. After similar preprocessing, we applied GridSearchCV to compute the best parameters for AdaBoost.
      i. Best Hyperparameters: {'learning_rate': 1.0, 'n_estimators': 100}
      ii. Best Accuracy Score: 0.8852167109550134
      iii. Test Accuracy: 0.886724104170746

   b. For feature selection, we used Recursive Feature Elimination, Cross-Validated (RFECV) feature selection with AdaBoost Classifier (along with the best parameters computed before). It selected 15 features and gave a ranking of features as well.

8. **Linear & Polynomial Regression**

   a. The pre-processing remains the same, which we have mentioned in the preprocessing section, involving checking for null values, removing duplicates, removing outlier points, and encoding the categorical columns.
   b. We used polynomial regression for multiple values of n, and L1 and L2 regularisation. The final predictions were mapped to 1 if the values were greater than 0.5 and the others to 0.
   c. As expected, the linear regression was not as accurate, with a polynomial regression of order 3 giving the highest accuracy of 0.0996 on the validation data.

d. SelectKBest was used for feature selection giving 13 of the best features, which led to a slight increase in accuracy to 0.1014.
e. However, since we did binary classification, the results vary a lot for both the test and validation data, which proves the fact that Linear Regression cannot be used for a classification problem.

Discussion on the Performance of Different Approaches:

Our goal was to develop models for predicting loan default risk, targeting the binary variable 'Default' (0 or 1). We explored several models, each with unique strengths:

1. **XGBoost**: This model achieved the highest accuracy due to its ability to handle complex patterns and interactions, with optimal tuning of parameters like learning rate and depth through GridSearchCV.
2. **Gradient Boosting & AdaBoost**: Both ensemble methods provided competitive accuracy, with RFECV-based feature selection improving model focus. However, they did not reach XGBoost’s precision, partially due to their insufficient feature handling.
3. **Linear and polynomial Regression**: As a baseline, binary classification was created by thresholding predictions at 0.5. However, the linear model was too simplistic, capturing only limited relationships.

Our interpretation of why XGBoost Gave the Highest Accuracy:

XGBoost excelled due to the following factors:

1. **Optimal Feature Selection**: XGBoost’s feature importance metrics led us to drop 'LoanTerm,' which improved performance by reducing noise.
2. **Effective Hyperparameter Tuning**: Tuning with GridSearchCV allowed XGBoost to fit complex patterns without overfitting.
3. **Robustness to Noise**: XGBoost’s regularisation capabilities and iterative boosting made it resilient and precise in handling complex data patterns.

Reference:
https://www.krayonnz.com/user/doubts/detail/623b2b7235e21e005f953106/what-are-the-advantages-and-disadvantages-of-XGBoost

Interesting Observations:

- **Surprising Performance of Linear Regression**: Initially, we anticipated a low accuracy with Linear Regression, given its unsuitability for binary classification. In validation, it indeed yielded low accuracy (0.101), but surprisingly, the test data accuracy reached 0.884. This result may be due to threshold adjustments for binary classification and the high number of “0” values in our extensive test set, which may have influenced the outcome.

- **Expectations from Common Classification Models**: Typically, models like KNNs, Decision Trees, and Random Forests are expected to perform well for binary classification problems like this one. However, XGBoost consistently outperformed all other models
- **Importance of Feature Selection and Engineering**: By using feature importance techniques and dropping the “LoanTerm” feature for XGBoost, we saw a clear boost in model accuracy. This finding highlights how feature engineering can significantly influence model performance and how some features might add noise rather than predictive value.
- **Effectiveness of Hyperparameter Tuning with GridSearchCV**: Models where GridSearchCV was applied, especially in tuning parameters for XGBoost, GradientBoost, Decision Tree, Random forest, etc. showed substantial improvements in accuracy. This underscores the critical role of hyperparameter optimization in enhancing model outcomes.

References:

- https://analyticsindiamag.com/topics/types-of-classification-model/
- https://www.geeksforgeeks.org/xgboost/