Machine Learning Project Report: Credit Score Classification

Team HIV
Harsh Vardhan Singh IMT2022101
Ishan Singh IMT2022124
Vansh Sinha IMT2022122

14th November, 2024

1 Problem Statement

As data scientists at a global finance company, we were tasked with developing an intelligent system to categorize individuals into specific credit score brackets. By leveraging a rich dataset of financial and credit-related information, our objective was to minimize manual intervention in credit evaluation by creating a machine learning model capable of accurately classifying individuals based on their credit rank bracket.

2 Summary of Dataset

The dataset contains financial and credit-related information such as salary, bank accounts, credit utilization, debt, and payment behavior. These attributes provide valuable insights into an individual's financial health and creditworthiness, which form the foundation for classifying credit score brackets.

3 Issues with Dataset

3.1 Dirty Data

- **Completion Issues** like filling null values in columns.
- **Validity Issues** like non-numeric values in numerical columns, wrong data-type assigned to columns and impossible values in some columns.
- **Accuracy Issues** like duplicate entries.

3.2 Messy Data

- **Unnecessary Columns** which are not useful for model training.
- **Transformations** needed for some columns such as one-hot encoding.

4 Data Preprocessing

4.1 Data Cleaning

- Columns such as *ID*, *Customer_ID*, *Month*, *Name*, and *Number* were deemed irrelevant for prediction and excluded.
- *Base_Salary_PerMonth* contains missing values, which were imputed using the average value for each *Customer_ID*.
- *Age* contains non-numeric, negative, and extremely large values, requiring data correction to ensure validity.
- *Credit_History_Age* has missing values and requires transformation into a consistent, usable format.

- Columns like *Profession*, *Income_Annual*, *Current_Debt_Outstanding*, and *Credit_Limit* contain non-numeric or invalid entries that were addressed during preprocessing.
- *Credit_Mix* has null values in the form of underscores (_), which were corrected or imputed.
- *Payment_Behaviour* and *Monthly_Balance* contain dirty values that were standardized.
- *Total_Delayed_Payments* contains extremely large values, which were capped or corrected to ensure data consistency.
- *Rate_Of_Interest* has outliers with unrealistically high values, which were addressed to avoid skewing the model.
- Negative values in columns such as *Total_Bank_Accounts* were replaced with valid approximations or dropped.

4.2 Column Selection and Exclusion

- Columns such as *Total_Current_Loans* were dropped in favor of more informative variables like *Loan_Type*.
- *Loan_Type* was split into distinct columns based on its comma-separated values.
- Columns that introduce redundancy or are irrelevant for prediction, such as *Customer_ID* and *Number*, were excluded.

4.3 Outlier Handling

- For each feature under consideration, the first quartile (Q1, 25th percentile) and third quartile (Q3, 75th percentile) were calculated.
- The IQR, defined as the difference between Q3 and Q1, was used to establish lower and upper thresholds:

$$\text{Lower Bound} = Q1 - 1.5 \times \text{IQR}, \quad \text{Upper Bound} = Q3 + 1.5 \times \text{IQR}$$

- Values below the lower bound were capped at the lower threshold, and values above the upper bound were capped at the upper threshold.
- The same thresholds derived from the training data were applied to the test data to ensure consistency.
- Visualizations such as boxplots were generated to compare the distribution of data before and after capping, ensuring the method effectively reduced outliers without significantly altering the underlying data structure.

4.4 Encoding

- One-Hot Encoding (OHE) was applied to categorical columns such as *Profession* and *Payment_Behaviour* to ensure compatibility with the model.
- Null or invalid values in *Payment_Behaviour* were cleaned prior to encoding.

5 Modeling and Experiments

5.1 Models Tested

We tested the following classifiers to evaluate their performance on the dataset:

- **DecisionTreeClassifier:** A simple tree-based model capable of capturing non-linear relationships in the data.
  - Used with **EasyEnsembleClassifier** for imbalanced learning.

  - Applied with **Random Oversampling** for balancing the dataset.
  - Applied with **Random Undersampling** for balancing the dataset.
  - Combined with **SMOTE** for synthetic oversampling.
  - Combined with **ADASYN** for adaptive synthetic oversampling.
  - Combined with **Borderline SMOTE** for enhanced boundary sampling.

- **KNeighborsClassifier:** A distance-based algorithm useful for capturing local patterns in the data.
  - Utilized the **EasyEnsembleClassifier** to effectively address class imbalance in the dataset.
  - Applied with **Random Oversampling** for balancing the dataset.
  - Applied with **Random Undersampling** for balancing the dataset.
  - Combined with **SMOTE** for synthetic oversampling.
  - Combined with **ADASYN** for adaptive synthetic oversampling.
  - Combined with **Borderline SMOTE** for enhanced boundary sampling.

- **RandomForestClassifier:** An ensemble method combining multiple decision trees to improve generalization and reduce overfitting.
  - Utilized the **EasyEnsembleClassifier** to effectively address class imbalance in the dataset.
  - Applied with **Random Oversampling** for balancing the dataset.
  - Applied with **Random Undersampling** for balancing the dataset.
  - Combined with **SMOTE** for synthetic oversampling.
  - Combined with **ADASYN** for adaptive synthetic oversampling.
  - Combined with **Borderline SMOTE** for enhanced boundary sampling.

- **AdaBoostClassifier:** A boosting algorithm that iteratively focuses on misclassified instances to improve accuracy.
  - Utilized the **EasyEnsembleClassifier** to effectively address class imbalance in the dataset.
  - Applied with **Random Oversampling** for balancing the dataset.
  - Applied with **Random Undersampling** for balancing the dataset.
  - Combined with **SMOTE** for synthetic oversampling.
  - Combined with **ADASYN** for adaptive synthetic oversampling.
  - Combined with **Borderline SMOTE** for enhanced boundary sampling.

- **XGBClassifier:** An efficient implementation of gradient boosting optimized for structured datasets.
  - Utilized the **EasyEnsembleClassifier** to effectively address class imbalance in the dataset.
  - Applied with **Random Oversampling** for balancing the dataset.
  - Applied with **Random Undersampling** for balancing the dataset.
  - Combined with **SMOTE** for synthetic oversampling.
  - Combined with **ADASYN** for adaptive synthetic oversampling.
  - Combined with **Borderline SMOTE** for enhanced boundary sampling.

- **MultinomialNB:** Not tested due to incompatibility with negative values.
- **BernoulliNB:** Not tested due to incompatibility with negative values.
- **GaussianNB:** Not tested due to incompatibility with negative values.
- **LogisticRegression:** Not utilized in favor of its inclusion in Checkpoint 2.

5.2 Model Performance and Evaluation

To evaluate the performance of various classifiers, we employed a systematic approach using stratified K-Fold cross-validation. This method ensured that the class distribution in the target variable was maintained across all training and testing splits, providing a robust evaluation framework. The steps undertaken were as follows:

- **Dataset Preparation:** The feature set (X) was created by dropping the target column *Credit_Score* from the training data, while the target variable (y) consisted of the *Credit_Score* column.
- **Cross-Validation Setup:** We initialized a *StratifiedKFold* cross-validator with 10 splits to ensure balanced class representation in each fold.
- **Model Evaluation:** Each classifier was evaluated using the *cross_validate* function, with the *F1 Score (Macro)* as the primary scoring metric. This metric was chosen to account for class imbalance by averaging F1 scores across all classes.
- **Parallel Computation:** To optimize computational efficiency, the cross-validation process was executed in parallel across multiple jobs using the *n_jobs=-1* parameter.
- **Result Aggregation:** For each classifier, the mean F1 Score (Macro) across all folds was calculated and recorded. These scores provided a comprehensive measure of the model's ability to balance precision and recall across all credit score brackets.
- **Performance Tracking:** The results were stored in a DataFrame, with each row representing a classifier and its corresponding mean F1 Score (Macro). This allowed for easy comparison of model performance.

This structured evaluation process ensured a fair and consistent comparison of all classifiers while addressing the challenges posed by class imbalance in the dataset.

5.3 Model Selection and Justification

The **Random Forest Classifier** was selected as the final model due to its consistently superior performance across various sampling techniques like *SMOTE*, *EasyEnsemble*, *Random Oversampling*, etc. It achieved the highest F1 Score (Macro) in almost all configurations, demonstrating robust handling of class imbalance and excellent generalization. Its ensemble approach and interpretability further solidified its selection for deployment.

5.4 Hyperparameter Tuning

- **Random Forest Variants:** Multiple variants of the Random Forest classifier were evaluated, each combined with different sampling techniques to address class imbalance. The variants included:
  - *Normal_RF*: Standard Random Forest without resampling.
  - *Balanced_RF*: Random Forest with balanced class weights.
  - *Over*: Random Forest with Random Oversampling.
  - *Under*: Random Forest with Random Undersampling.
  - *SMOTE*: Random Forest with SMOTE (Synthetic Minority Oversampling Technique).
  - *ADASYN*: Random Forest with ADASYN for adaptive synthetic sampling.
  - *BorderlineSMOTE*: Random Forest with Borderline SMOTE for enhanced boundary sampling.
- **Parameter Grid:** The following hyperparameters were tuned to optimize model performance:
  - *n_estimators*: Set to 200 to control the number of trees in the forest.
  - *ccp_alpha*: Pruning parameter values were explored over a range from 10^{-6} to 10^{-4} to balance model complexity and performance.
- **GridSearchCV for Optimization:**

- *GridSearchCV* was used to perform exhaustive search over the specified parameter grid.
- A *StratifiedKFold* cross-validator was used with 10 splits to maintain class distribution across folds.
- The *f1_macro* metric was used as the scoring criterion to optimize for balanced performance across all classes.
- Parallel processing (*n_jobs=-1*) was employed to improve computational efficiency.

6 Additional Insights

6.1 Imbalance of Target Variable

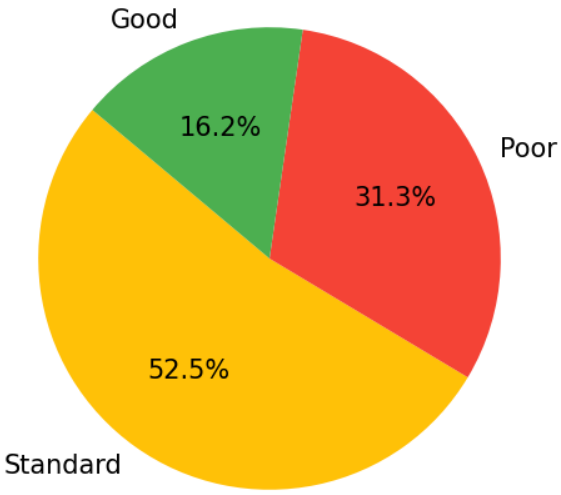

Figure 1: Distribution of Credit Score

- The credit score distribution is imbalanced, with 52.5% classified as `Standard`, 31.3% as `Poor`, and only 16.2% as `Good`, indicating that moderate and low creditworthiness are far more common than high creditworthiness.

The credit score distribution shows significant imbalance, with a dominant `Standard` category, a substantial `Poor` group, and a smaller `Good` group. Addressing this imbalance requires techniques such as oversampling (e.g., `SMOTE`, `ADASYN`), undersampling, or ensemble methods (e.g., `EasyEnsemble`) to prevent classifier bias toward the majority class and enhance balanced performance across categories. This approach improves the model's accuracy in identifying both high and low creditworthiness.

6.2 Payment Behaviour Impact on Credit Score

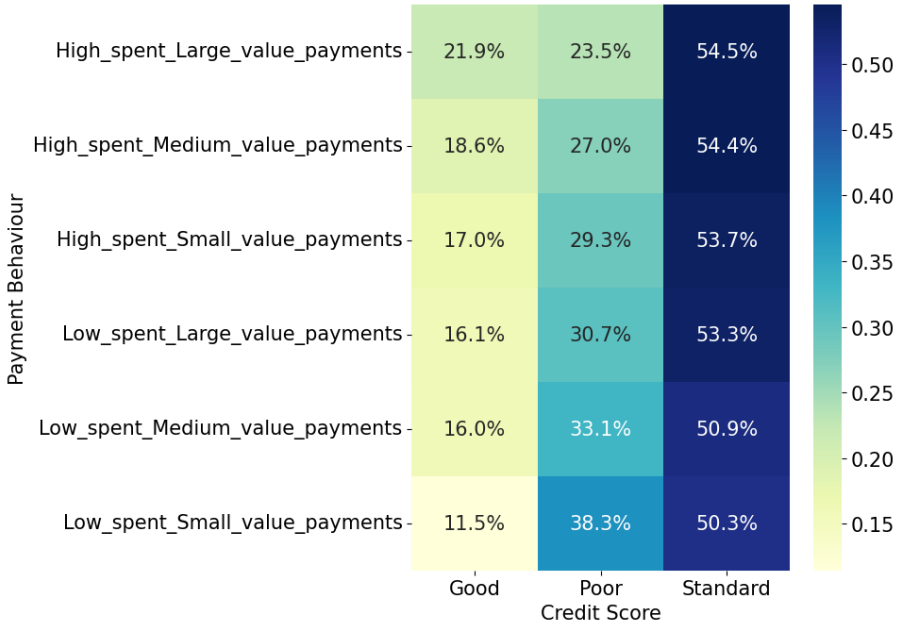

Figure 2: Heatmap of Payment Behaviour and Credit Score

- **Good Credit Scores with High Spending** High spending is associated with `Good` credit scores (e.g., 21.9% for `High_spent_Large_value_payments`), though they remain a minority.
- **Poor Credit Scores with Low Spending** Low spending patterns, such as `Low_spent_Small_value_payments` (38.3%), are linked to a higher share of `Poor` credit scores.
- **High Spending and Poor Credit** High spending does not guarantee good credit, as significant proportions of high spenders (e.g., 23.5% in `High_spent_Large_value_payments`) have `Poor` credit scores.

6.3 Credit Mix Impact on Credit Score

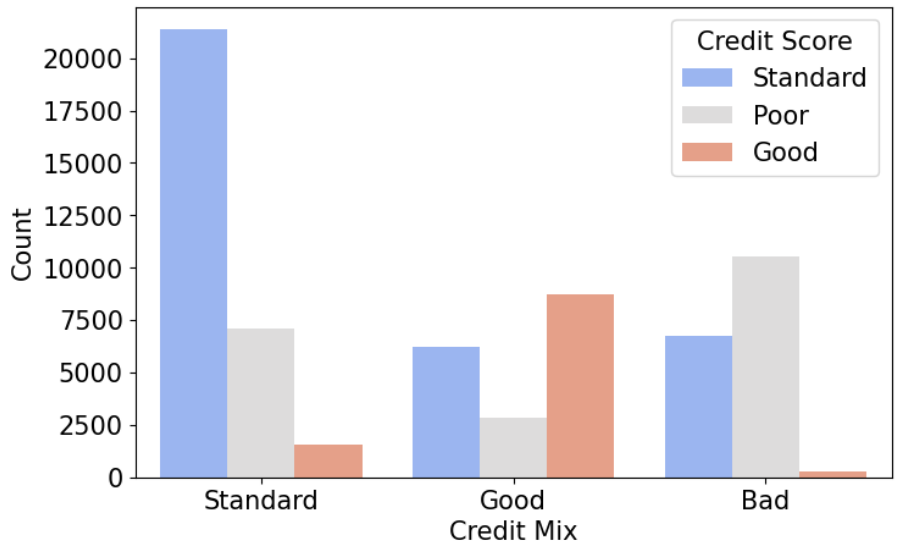

Figure 3: Bar Plot of Credit Mix grouped by Credit Score

- **Good Credit Mix**: Mostly linked to *good credit scores*, suggesting that a balanced credit profile often aligns with higher creditworthiness.
- **Bad Credit Mix**: Primarily linked to *poor credit scores*, implying that this mix is generally associated with lower creditworthiness.

Credit mix has a significant impact on credit scores, with good mixes correlating with higher scores and bad mixes with lower scores.