Machine Learning Checkpoint-1

Lend or Lose Report

Team Members:

- Krishna Sathwik-IMT2022045
- Pudi Kireeti-IMT2022059
- Srinivas Acharya-IMT2022066

Git Hub link:
https://github.com/krishnasathwik/ML_Project

Loan Default Prediction

Project Overview

This project aimed to predict loan defaults using historical data and portfolio management. In our approach, we explored multiple classification models, including Decision Trees, XGBoost (with GridSearchCV for tuning), Logistic Regression and HistGradientBoostingClassifier (using RandomSearchCV). After evaluating model performance, HistGradientBoostingClassifier emerged as the best performer, offering superior accuracy and stability on this dataset. Accuracy was (88.807) on the given Train Data set.

Data Preprocessing

1. **Data Loading and Initial Cleaning**
    a. **Loading Data**: We loaded both training and testing datasets from CSV files (train.csv and test.csv). Initial inspection included verifying column consistency, datatype correctness, and ensuring no immediate data issues.We validated the absence of any missing values in both datasets, establishing a clean base for further preprocessing steps.

   b. **Duplicate Removal**: To enhance data quality, we identified and removed duplicate entries from the training dataset. Duplicate data can introduce redundancy, reduce model accuracy, and skew the training process, so this step was crucial to ensure a reliable foundation.
      Duplicates were not present in our dataset
   c. **LoanID Column Removal**: Since LoanID functions solely as an identifier without predictive value, it was removed from both datasets. Retaining it could introduce noise, distract from relevant features, and complicate the learning process.
   d. **Bias identification:** we identified that there is a bias of ratio 9:1 in the given training dataset. That is for every 9 one's in the dataset there is only 1 zero. The dataset is more biased towards one's.

2. **Outlier Handling**
   a. **Outlier Detection and Strategy**: Outliers can distort model interpretation, especially in regression and classification trees. We applied interquartile range (IQR) methodology, extreme values in each numerical column to reduce the influence of outliers without altering the data distribution.
      Outliers were not found in our dataset

   b. **Columns Processed**: Outlier handling was conducted for columns with continuous values, including Age, Income, LoanAmount, CreditScore, MonthsEmployed, NumCreditLines, InterestRate, LoanTerm, and DTIRatio. This step improved data quality by reducing extreme data points that could disproportionately affect model predictions.
3. **Feature Engineering**
   a. **Categorical Encoding**: We mapped categorical variables to numerical values to facilitate model processing. For instance, values in MaritalStatus, Education, EmploymentType, HasMortgage, HasDependents, LoanPurpose, and HasCoSigner were converted using predefined mappings to maintain consistency across the datasets.
   b. **Combined Feature Creation**: To enrich the dataset, we created a new feature, Marital_Dependents, by combining MaritalStatus and HasDependents. This derived feature attempts to capture a potential interplay between marital status and the dependency load, providing insights into the applicant's overall dependency burden, which may correlate with repayment capacity and risk profile.
4. **Scaling and Standardization**
   a. **Sequential Scaling and Standardization**: We applied MinMax scaling to normalize features, followed by StandardScaler to ensure a zero mean and unit variance for each feature, a common best practice to facilitate

model convergence. Scaling and standardization are especially critical for gradient-based models like XGBoost and HistGradientBoosting, enabling more consistent and interpretable results.

b. **Encountering NaN Values**: During scaling and standardization, we encountered NaN values, despite no initial missing data. This phenomenon occurs due to identical feature values (e.g., if all values in a column are the same) or when extreme values reintroduce issues like division by zero, resulting in undefined values. These NaNs were subsequently handled through imputation.

NaN's were not found initially in our dataset
(No initial missing data)

5. **Handling NaN Values with KNN Imputer**
   a. **KNN Imputation for Missing Values**: To address NaN values introduced during scaling and standardization, we used KNN (K-Nearest Neighbors) imputation. KNN imputation fills NaN values by referencing the nearest available values within the dataset, ensuring continuity in data patterns and consistency across records. This imputation technique was effective in restoring data completeness without compromising the quality of transformed features.

Model Selection and Hyperparameter Tuning

1. **Model Exploration and Selection**
   a. We explored several models to identify the best performer, each chosen based on its suitability for classification tasks:
      i. **Decision Trees**: This model got an accuracy of 88.39% on validation
      data and 88.447% on test data. These are fast to implement and offer
      interpretable results but can overfit and lack robustness on complex data
      ii. **XGBoost**: This model got an accuracy of 88.63% on validation data and 88.776 % on test data. A powerful gradient-boosting model, XGBoost is popular for its scalability and feature importance capabilities. To enhance its effectiveness, we used GridSearchCV to
      fine-tune hyperparameters such as n_estimators, max_depth, learning_rate, and subsample.
      iii. **HistGradientBoostingClassifier**: This model got an accuracy of 88.51% on validation data and 88.807% on test data. This model, a

gradient-boosting variant, was selected for its computational efficiency on larger datasets and its adaptive approach to different types of feature distributions. Using RandomSearchCV allowed us to perform a broad parameter search, testing multiple configurations in a time-efficient manner. HistGradientBoostingClassifier achieved higher accuracy than Random Forest and other models by effectively handling high-dimensional data and optimizing performance through gradient boosting, which reduces bias and variance more efficiently

Iv. **Logistic Regression:** This model gave an accuracy of 88.41% on validation data and 88.584 % on Test data, this model was not the best model as it gave lesser accuracy on test data compared to HistGradientBoostingClassifier (88.807%). Logistic Regression assumes linear relationships between features and the target, which can limit its accuracy on complex data. HistGradientBoostingClassifier captures non-linear patterns and interactions, enhancing its predictive performance.

v. **Random Forest:** This model gave an accuracy of 88.34% on validation data and 88.470% on Test data, this model was not the best model as it gave lesser accuracy on test data compared to HistGradientBoostingClassifier (88.807%).

vi. **NaiveByes:** This model gave an accuracy of 88.39% on validation data and 88.558% on Test data. Naive Bayes assumes feature independence, which often doesn't hold in real-world data, thus reduces accuracy.

Vii. **Support Vector Machine:** This model got a validation accuracy of 88.39 % on validation data and 88.47% on test data. A powerful model, to enhance its effectiveness, we used GridSearchCV to fine-tune hyperparameters such as C,kernel and gamma.

viii. **Neural Networks:** This model got a validation accuracy of 88.7% on validation data and got accuracy of 88.77% on test dataset. A powerful unsupervised learning model where steps such as Forward pass,Back propagation,updation of weights take place and we have a certain number of hidden layers.

2. **Hyperparameter Tuning**
   a. **GridSearchCV and RandomSearchCV**: For XGBoost, we used GridSearchCV to meticulously examine combinations of specific hyperparameters, while for HistGradientBoostingClassifier, RandomSearchCV allowed us to evaluate a wider range of parameters in less time.
   b. **Key Parameters Considered**:
      i. **Number of Estimators**: Controls the number of boosting rounds.
      ii. **Max Depth**: Limits tree depth, which helps control overfitting.
      iii. **Learning Rate**: Adjusts the contribution of each tree to the final model.
      iv. **Subsample Rate**: Dictates the fraction of samples used for fitting each tree, enhancing generalization.
   c. **Results**: The HistGradientBoostingClassifier with RandomSearchCV delivered the best results, indicating an optimal balance of performance and interpretability.

Evaluation and Prediction

1. **Train-Test Split**
   a. **Training and Validation Sets**: We split the training data into an 80% training and 20% validation set. This approach facilitated unbiased model evaluation and allowed us to fine-tune hyperparameters before final predictions on the test set.
2. **Final Model Training and Testing**
   a. **Best Model Selection**: With RandomSearchCV-tuned HistGradientBoostingClassifier as the chosen model, we retrained it on the full training set and validated its performance on hold-out validation data.
3. **Predictions and Submission Preparation**
   a. **Test Set Predictions**: Using the final trained model, we predicted Default probabilities on the test set after performing identical preprocessing steps to the training set.

b. **Submission File**: We generated a CSV file for submission containing LoanID and predicted Default values, ensuring it matched the required submission format.

Insights and Future Work

1. **Feature Engineering Effectiveness**
    a. Creating features like Marital_Dependents added value by combining individual characteristics into a more informative risk indicator. Additional feature engineering techniques, such as interaction terms or polynomial features, could be further explored.
2. **Handling Outliers and Scaling Challenges**
    a. Addressing outliers through capping and handling NaN values after scaling and standardization were crucial steps. The appearance of NaN values,

The exact reason for getting NaN values after standardization or normalization, despite having no missing values initially, is related to how these processes handle the input data mathematically.

The Exact Reason:

1. **Standardization (Zero Mean and Unit Variance)**:
    a. **Formula**: $Z=X-\mu/\sigma$
        i. X: Original value.
        ii. μ: Mean of the feature.
        iii. σ: Standard deviation of the feature.
    b. **Cause of NaN**:
        i. If a feature has **constant values** (all values are identical), the standard deviation (σ) becomes 0.
        ii. Division by zero is undefined, which results in NaN values for that feature during standardization.