Loan Approval Prediction Application Model Report

Problem Statement: Assume you are a loan risk officer at a large bank and you are tasked with determining whether a two-wheeler loan application will be accepted or rejected based on the data shared by the loan applicant and some additional data extracted about them from 3rd party.

Objective:

Build a classification model that can predict whether a loan application will be accepted or rejected based on the available data (both applicant-specific and third-party data).

Approach Taken: This section describes the methodology used to approach the problem, including data loading, exploration, preprocessing, and model selection.

Import necessary libraries

- pandas (pd) – For data manipulation and analysis.
- numpy (np) – For numerical operations and array handling.
- matplotlib (plt) – For creating static, animated, and interactive visualizations.
- seaborn (sns) – For statistical data visualization built on top of Matplotlib.
- scikit-learn (sklearn):
- train_test_split – For splitting datasets into training and validation sets.
- GridSearchCV – For hyperparameter tuning using cross-validation.
- StandardScaler – For standardizing features.
- LabelEncoder – For encoding categorical variables.
- RandomForestClassifier – For building the Random Forest classification model.
- accuracy_score – For calculating accuracy.
- confusion_matrix – For generating confusion matrix.
- classification_report – For detailed classification metrics.
- roc_auc_score – For calculating the ROC-AUC score.

Data Loading:

pd.read_csv is used to load the training and test datasets from CSV files into pandas DataFrames.

Assignment_Train.csv contains labeled data that will be used to train the model. It includes all relevant features and the target variable, i.e., "Application Status."

Assignment_Test.csv contains the same features as the training set but lacks the target variable, as it's the data on which the model's predictions will be tested.

pd.read_excel is used to load the Assignment_FeatureDictionary.xlsx, which provides details and descriptions of the features in the datasets. This helps you understand the significance of each feature for better model building.

2. Basic Dataset Inspection:

train_data.head()

- displays the first 5 rows of the training dataset. This is useful to quickly check if the data has been loaded correctly and to get an initial sense of the structure, values, and columns in the dataset.

train_data.info() provides a concise summary of the training dataset,

- Column names and their data types.
- Number of non-null values in each column, which helps identify missing data.
- Memory usage of the dataset.

train_data.describe() generates descriptive statistics for numeric columns. It includes:

- Count of non-null values.
- Mean, standard deviation, minimum, and maximum values.
- Percentiles (25%, 50%, 75%).

This is useful for gaining insights into the distribution of numerical data and identifying any potential data issues like extreme outliers.

Purpose:

- Data Loading: Load the datasets for training and testing.
- Initial Data Inspection: Understand the dataset structure, types of data, and the presence of missing values or outliers.

Exploratory Data Analysis (EDA):

Checking for Missing Values:

train_data.isnull().sum() checks for missing values in the dataset.

- **isnull()** returns a DataFrame of the same shape as train_data, where each element is True if the corresponding value in the dataset is NaN (missing) and False otherwise.
- **sum()** aggregates these True/False values for each column to show the total number of missing values in each column.

This step is crucial because missing values can negatively impact model training and prediction accuracy. The results will inform any necessary preprocessing steps, like imputing missing data or dropping incomplete records.

Visualizing Target Distribution (Application Status):

sns.countplot() is a function from the Seaborn library used to create a count plot, showing the frequency of different categories in the "Application Status" column.

- **x='Application Status'** specifies that the count of each unique value in the "Application Status" column should be plotted.
- **data=train_data** defines the DataFrame from which the data is drawn.

The count plot is used here to visualize the distribution of accepted vs. rejected loan applications, providing insight into whether the dataset is balanced or imbalanced. If the target classes are imbalanced (i.e., one class appears significantly more than the other), you may need to address this issue before training a model (e.g., using techniques like oversampling or undersampling).

3. Visualizing the Correlation Matrix:

- **train_data.corr()** computes the Pearson correlation matrix for the numeric columns in the dataset.
- **The correlation matrix** shows the pairwise correlation coefficients between the features. A value close to +1 or -1 indicates a strong linear relationship, while a value close to 0 indicates no linear relationship.
- **sns.heatmap()** creates a heatmap to visualize the correlation matrix.
- **annot=True** ensures that the actual correlation values are displayed on the heatmap.
- **fmt='.2f'** formats the correlation values to two decimal places.
- **cmap='coolwarm'** applies the color scheme to represent positive and negative correlations with different colors (e.g., cool for negative, warm for positive correlations).
- The **correlation heatmap** is useful for understanding relationships between features, identifying highly correlated variables (which might lead to multicollinearity), and selecting important features for model building.

Data Preprocessing:

Converting 'Date' Column to Datetime:

- **Purpose:**
  This block converts the 'Date' column in the dataset to a datetime format. Handling dates correctly is crucial when working with time-sensitive data (such as loan applications) because it allows for temporal analysis, time-based feature extraction, or trend analysis over time.
- **Error Handling:**
  The use of the try-except block ensures that if the 'Date' column is absent (raising a KeyError), the program doesn't crash. Instead, it prints an error message and displays

the available columns for debugging. This makes the code robust, especially when working with varying datasets.

- **Importance of Datetime Conversion**:
  Converting a date field into datetime allows operations like calculating the time difference, resampling, or extracting specific components like year, month, or day. For time-series data, this is essential.

Extracting Numerical Features:

- **Purpose:**
  This line selects all the columns in the dataset that are numerical (e.g., integer and float data types). Since correlation analysis is performed between numerical variables, this step ensures only relevant features are considered.
- **Why Select Numerical Data:**
  Performing correlation analysis on categorical or textual data can result in meaningless or erroneous values, so the function select_dtypes(include=['number']) filters out such columns. This ensures only numerical features are passed for analysis.
- **Context:**
  In a loan approval dataset, numerical columns might include applicant income, loan amount, age, or credit score. Understanding the correlation between these variables can reveal patterns useful for prediction.

Visualizing Correlation Matrix:

Purpose:

The goal here is to visualize the correlation between the numerical variables using a heatmap. This allows us to detect relationships or patterns in the dataset that are not immediately apparent.

Steps:

- **numerical_features.corr():** This function computes the Pearson correlation coefficient for each pair of numerical features. The correlation coefficient tells us how much two variables are linearly related. It ranges from:
- **+1:** Strong positive correlation (when one variable increases, the other also increases).
- **-1:** Strong negative correlation (when one variable increases, the other decreases).
- **0:** No linear relationship.
- **sns.heatmap():** This function creates a heatmap that visualizes these correlations:
- **annot=True:** This parameter displays the correlation coefficients directly on the heatmap.
- **fmt='.2f'**: It ensures the values are shown with two decimal precision.

- **cmap='coolwarm':** This color map uses a gradient to represent positive and negative correlations, with blue for negative and red for positive.
- **Heatmap Interpretation:**
- The heatmap visually displays which features are strongly correlated with each other.
- This is useful in identifying redundant features (highly correlated with others) or features with little correlation to the target variable.

Benefits of Correlation Analysis:

- **Multicollinearity:** Detecting variables that are highly correlated with each other (e.g., income and loan amount) helps avoid multicollinearity in machine learning models, which can lead to inaccurate predictions.
- **Feature Selection:** Identifying the most relevant features that correlate with the target variable (e.g., loan approval) helps improve model accuracy and reduces the complexity of the model.
- **Pattern Recognition:** Strong correlations can reveal hidden relationships in the data, which can be crucial for making data-driven decisions.

Data Standardization:

- The **StandardScaler()** is used to standardize the features by removing the mean and scaling to unit variance. This is often essential for models like Random Forest to ensure that all features contribute equally to the model performance.
- **X_train_scaled and X_val_scaled** store the standardized versions of the training and validation sets, ensuring that both have the same scaling.

Random Forest Model:

- The **RandomForestClassifier** is a machine learning model that operates by constructing multiple decision trees during training and outputting the mode of the classes for classification.
- The model is trained on the standardized training data using the fit() function.

Making Predictions:

The model makes predictions on the validation data (X_val_scaled) using the **predict()** method.

Model Evaluation:

The evaluation of the model is done using several metrics:

- **Accuracy:** Measures the percentage of correct predictions.

- **Confusion Matrix:** Displays the true positives, true negatives, false positives, and false negatives to give a clear idea of where the model is making errors.
- **Classification Report:** Provides detailed metrics like precision, recall, and F1-score, which further measure the model's effectiveness in classification tasks.

Label Encoding Categorical Features:

- `LabelEncoder()` is used to convert categorical variables into numerical values, which are required by most machine learning models.
- First, the target variable `'Application Status'` is encoded using `fit_transform()` before handling missing values.

Handling Missing Values:

- For numeric columns, missing values are replaced by the column mean using `fillna()`.
- For non-numeric columns (categorical), missing values are filled with the most frequent value (mode).

Handling the Test Data:

- Similar encoding and missing value treatment is applied to the test dataset.
- Categorical values in both `train_data` and `test_data` are encoded using the combined data to ensure consistency between the datasets.

Numeric Conversion:

- Columns that can be converted to numeric types are handled using `pd.to_numeric()`. For those that cannot (likely categorical), they are filled with appropriate values based on their type.

Feature Importance Calculation:

- The feature_importances_ attribute of the RandomForestClassifier returns the importance scores of each feature based on how useful they were in making predictions.
- These scores represent the contribution of each feature to the model’s predictive power.

Ranking Features:

- The features are ranked by importance by sorting the importance scores in descending order using np.argsort(importances)[::-1].

- The ranked features are printed along with their respective importance values, giving insight into which features are most relevant in determining loan application outcomes.

Plotting Feature Importances:

- A bar chart is plotted to visually represent the importance of each feature. The x-axis represents the features, and the y-axis represents the importance scores.
- This plot helps in easily identifying the most influential features that impact model performance.

Significance:

- Understanding feature importance allows you to identify which features significantly affect the model's decision-making process. It can also help in feature selection, model interpretation, and improving model accuracy by removing irrelevant features.

Insights and Conclusions from Data :

- **Key Features Identified:** The most important features influencing loan approval are highlighted, helping focus on key factors like income, credit score, or employment status.
- **Data-Driven Decisions:** Insights can guide the bank in refining loan policies, emphasizing critical factors to reduce risk.
- **Model Transparency:** The feature importance shows how the model makes predictions, making it easier to explain decisions to stakeholders.
- **Improvement Areas:** Low-importance features can be improved through feature engineering or removed to simplify the model.
- **Data Quality:** The analysis underscores the need for quality data and well-engineered features to improve predictions.

Performance on Train Dataset :

- **Accuracy:** Shows the overall correctness of the model on the training data.
- **Confusion Matrix:** Reveals the counts of true positives, true negatives, false positives, and false negatives.
- **Classification Report:** Provides metrics like precision, recall, and F1-score for evaluating model performance on each class.
- **Model Fit:** Indicates how well the model fits the training data, but validation is needed to ensure it generalizes well.