**Loan Status Prediction**

**Project Overview**

The goal of this project is to develop a machine learning model that predicts the loan status (approved or rejected) based on various applicant features. The project uses a dataset containing historical loan data to train and evaluate different classification models.

**Dataset**

The dataset consists of two CSV files:

**train\_loan\_data.csv:** Contains the training data with applicant features and corresponding loan status.

**test\_loan\_data.csv:** Contains the test data with applicant features for which loan status needs to be predicted.

The dataset is loaded using the pandas library:

train\_data = pd.read\_csv('train\_loan\_data.csv')

test\_data = pd.read\_csv('test\_loan\_data.csv')

**Dependencies**

The project requires the following libraries:

**pandas:** For data manipulation and analysis.

**numpy:** For numerical operations.

**matplotlib:** For data visualization.

**seaborn:** For enhanced data visualization.

**scikit-learn:** For machine learning algorithms and evaluation metrics.

These libraries are imported at the beginning of the script:

**import pandas as pd**

**import numpy as np**

**import matplotlib.pyplot as plt**

**import seaborn as sns**

**from sklearn.model\_selection import train\_test\_split, GridSearchCV**

**from sklearn.preprocessing import StandardScaler, OneHotEncoder**

**from sklearn.impute import SimpleImputer**

**from sklearn.pipeline import Pipeline**

**from sklearn.compose import ColumnTransformer**

**from sklearn.linear\_model import LogisticRegression**

**from sklearn.tree import DecisionTreeClassifier**

**from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier**

**from sklearn.metrics import f1\_score, classification\_report**

**Project Structure**

The project follows the following structure:

**Exploratory Data Analysis (EDA):**

The shape, info, and description of the training and test datasets are analyzed using pandas methods:

**print("Training data shape:", train\_data.shape)**

**print("Test data shape:", test\_data.shape)**

**print("\nTraining data info:")**

**train\_data.info()**

**print("\nTest data info:")**

**test\_data.info()**

**print("\nTraining data description:")**

**train\_data.describe()**

The distribution of the target variable (loan status) is visualized using a countplot from seaborn:

**plt.figure(figsize=(8, 6))**

**sns.countplot(x='loan\_status', data=train\_data)**

**plt.title('Loan Status Distribution')**

**plt.show()**

**Data Preprocessing:**

The dataset is split into numeric and non-numeric features:

**numeric\_features = train\_data.select\_dtypes(include=[np.number]).columns**

**non\_numeric\_features = train\_data.select\_dtypes(exclude=[np.number]).columns**

The 'loan\_status' column is removed from non-numeric features for both training and test data:

**non\_numeric\_features\_train = non\_numeric\_features.drop('loan\_status')**

**non\_numeric\_features\_test = non\_numeric\_features.drop('loan\_status')**

Missing values are imputed separately for numeric and non-numeric features using the SimpleImputer class:

**imputer\_numeric = SimpleImputer(strategy='median')**

**imputer\_non\_numeric = SimpleImputer(strategy='most\_frequent')**

**train\_data[numeric\_features] = imputer\_numeric.fit\_transform(train\_data[numeric\_features])**

**train\_data[non\_numeric\_features\_train] = imputer\_non\_numeric.fit\_transform(train\_data[non\_numeric\_features\_train])**

**test\_data[numeric\_features] = imputer\_numeric.transform(test\_data[numeric\_features])**

**test\_data[non\_numeric\_features\_test] = imputer\_non\_numeric.transform(test\_data[non\_numeric\_features\_test])**

**Feature Engineering and Selection:**

This stage is not implemented in the provided code but is mentioned as a placeholder for future improvements.

**Model Training and Evaluation:**

The training data is split into features (X\_train) and target (y\_train):

**X\_train = train\_data.drop('loan\_status', axis=1)**

**y\_train = train\_data['loan\_status']**

**X\_test = test\_data**

Preprocessing pipelines are created for numeric and categorical features using StandardScaler and OneHotEncoder, respectively:

**numeric\_transformer = Pipeline(steps=[**

**('scaler', StandardScaler())**

**])**

**categorical\_transformer = Pipeline(steps=[**

**('onehot', OneHotEncoder(handle\_unknown='ignore'))**

**])**

**preprocessor = ColumnTransformer(**

**transformers=[**

**('num', numeric\_transformer, numeric\_features),**

**('cat', categorical\_transformer, non\_numeric\_features\_train)**

**])**

**Multiple classification models are defined in a dictionary:**

**models = {**

**'Logistic Regression': LogisticRegression(),**

**'Decision Tree': DecisionTreeClassifier(),**

**'Random Forest': RandomForestClassifier(),**

**'Gradient Boosting': GradientBoostingClassifier()**

**}**

Each model is trained using a pipeline that combines the preprocessor and the model itself. The models are evaluated using the F1-score on a validation set:

**best\_model = None**

**best\_score = 0**

**for name, model in models.items():**

**print(f"\nTraining {name} model...")**

**pipeline = Pipeline(steps=[**

**('preprocessor', preprocessor),**

**('model', model)**

**])**

**X\_train\_split, X\_val, y\_train\_split, y\_val = train\_test\_split(X\_train, y\_train, test\_size=0.2, random\_state=42)**

**pipeline.fit(X\_train\_split, y\_train\_split)**

**y\_pred = pipeline.predict(X\_val)**

**score = f1\_score(y\_val, y\_pred, average='weighted')**

**print(f"{name} F1-score: {score:.4f}")**

**if score > best\_score:**

**best\_score = score**

**best\_model = pipeline**

The best-performing model is selected based on the highest F1-score.

**Hyperparameter Tuning:**

Hyperparameter tuning is performed for the best-performing model using GridSearchCV:

**if isinstance(best\_model.named\_steps['model'], RandomForestClassifier):**

**param\_grid = {**

**'model\_\_n\_estimators': [100, 200, 300],**

**'model\_\_max\_depth': [None, 5, 10],**

**'model\_\_min\_samples\_split': [2, 5, 10]**

**}**

**elif isinstance(best\_model.named\_steps['model'], GradientBoostingClassifier):**

**param\_grid = {**

**'model\_\_n\_estimators': [100, 200, 300],**

**'model\_\_learning\_rate': [0.01, 0.1, 0.5],**

**'model\_\_max\_depth': [3, 5, 7]**

**}**

**elif isinstance(best\_model.named\_steps['model'], LogisticRegression):**

**param\_grid = {**

**'model\_\_C': [0.1, 1, 10],**

**'model\_\_penalty': ['l2']**

**}**

**else:**

**param\_grid = {}**

**if param\_grid:**

**grid\_search = GridSearchCV(best\_model, param\_grid, cv=5, scoring='f1\_weighted', n\_jobs=-1)**

**grid\_search.fit(X\_train, y\_train)**

**print("\nBest hyperparameters:")**

**print(grid\_search.best\_params\_)**

**best\_model = grid\_search.best\_estimator\_**

The best hyperparameters are determined using cross-validation, and the best model is updated with the tuned hyperparameters.

**Prediction and Saving Results:**

**The best model is used to make predictions on the test data:**

**y\_test\_pred = best\_model.predict(X\_test)**

**The predicted loan status is saved to a CSV file named 'test\_predictions.csv':**

**output = pd.DataFrame({'loan\_status': y\_test\_pred})**

**output.to\_csv('test\_predictions.csv', index=False)**

**Results and Insights**

The script trains and evaluates multiple classification models, including Logistic Regression, Decision Tree, Random Forest, and Gradient Boosting. The performance of each model is assessed using the F1-score, which provides a balanced measure of precision and recall.

**The results obtained from training and evaluating the models are as follows:**

Logistic Regression F1-score: 0.7472

Training Decision Tree model...

Decision Tree F1-score: 0.7330

Training Random Forest model...

Random Forest F1-score: 0.7251

Training Gradient Boosting model...

Gradient Boosting F1-score: 0.7303

Based on these results, the **Logistic Regression model achieves the highest F1-score of 0.7472,** outperforming the other models. Therefore, Logistic Regression is selected as the best model for loan status prediction on this dataset.

**Best model: LogisticRegression**

**Best F1-score: 0.7472**

The best model is then subject to hyperparameter tuning using GridSearchCV to further optimize its performance. The tuned model is used to make predictions on the test data.

It's important to note that the performance of the models may vary depending on the specific characteristics of the dataset and the hyperparameters used. The Logistic Regression model has proven to be the most effective for this particular dataset, achieving the highest F1-score.

Further analysis and experimentation can be conducted to gain more insights into the model's performance, such as examining the confusion matrix, precision, recall, and ROC curve. Additionally, exploring feature engineering techniques and feature selection methods may help in improving the model's performance and interpretability.

Overall, the Logistic Regression model has demonstrated its effectiveness in predicting loan status based on the given applicant features, providing a solid foundation for making informed loan approval decisions.

**Usage**

* Ensure that the required libraries are installed.
* Place the 'train\_loan\_data.csv' and 'test\_loan\_data.csv' files in the same directory as the script.
* Run the script.
* The script will perform EDA, preprocess the data, train and evaluate models, tune hyperparameters, and make predictions on the test data.
* The predicted loan status will be saved in the 'test\_predictions.csv' file.
* Results and Insights
* The script trains and evaluates multiple classification models, including Logistic Regression, Decision Tree, Random Forest, and Gradient Boosting.
* The best-performing model is selected based on the F1-score.
* Hyperparameter tuning is performed for the best model to optimize its performance.
* The tuned model is used to make predictions on the test data.

**Future Improvements**

* Implement feature engineering techniques to create new relevant features.
* Explore feature selection methods to identify the most important features for loan status prediction.
* Experiment with additional models or ensemble techniques to further improve the prediction accuracy.
* Conduct more in-depth analysis of the model's performance, including confusion matrix, precision, recall, and ROC curve.
* Integrate the trained model into a user-friendly application or API for real-time loan status prediction.

**Conclusion**

This project demonstrates the process of building a machine learning model for loan status prediction. By following the steps of EDA, data preprocessing, model training and evaluation, hyperparameter tuning, and prediction, we can develop a robust model that can assist in making informed loan approval decisions.

The provided code serves as a starting point for the loan status prediction project and can be further enhanced and customized based on specific requirements and data characteristics.