CSE587A Data Intensive Computing - Project Phase 2 Report

Title: Data-Driven Loan Default Prediction and Financial Analysis

Team members: Juseung Lee, Venkata Amballa

1. Summary of Phase 1

1.1. Problem Statement

Loan defaults raise significant issues in the financial sector that affect lenders, borrowers, and the overall economic environment. It results in financial losses for lenders and impedes borrowers' financial stability. Moreover, loan defaults can also lead to a loss of confidence in the lending ecosystem and can disrupt credit access for individuals and businesses. The main objective of this project is to develop predictive models that can address these challenges by accurately predicting key aspects of the risk of loan issuance.

1.2. Data Source and Description

The loan default dataset used in this project is available on the Kaggle [1]. The loan default dataset provides a comprehensive overview of loan repayment patterns across various regions throughout 2019. This dataset is a significant resource for analyzing and understanding loan repayment behaviors. It includes several deterministic factors such as loan_purpose, loan_amount, property_value, and Credit_Score. In this dataset, the ‘Status’ feature indicates the loan status that value 1 identifies loan granted and value 0 identifies loan not granted. Using the data from the ‘Status’ feature, we can make assumptions about whether a loan applicant will default on the loan. For example, ‘Status=0’ indicates the loan is not sanctioned and the loan applicant will default on the loan.

1.3. Data Cleaning/Processing

- **Normalizing numerical features**

  We applied robust scaling by using ‘RobustScaler()’ to the numerical features in the dataset to maintain all the features in a consistent range.

1.4. Cleaned Data

Cleaned data has been saved to 'Loan_Default_cleaned.csv' in phase 1. We renamed 'Loan_Default_cleaned_downsampled.csv', and we added a new file named 'Loan_Default_cleaned_all.csv' that contains all cleaned data without downsampling.

1.5. Changes we've made

We made the following changes in the phase 1 code,

1. Total_units: We realized that total_units is a scaler, so we converted it to an integer.

2. Age: This value is one-hot encoded. We resolved this by considering this feature as an integer.

2. Algorithms/Visualization

We loaded our cleaned dataset 'Loan_Default_cleaned_downsampled.csv', and selected 'status' as a target variable. Then, we split the data into training and testing sets where the test size is 0.2 and the train size is 0.8, using train_test_split. The metrics we have considered are **accuracy** and **F- β Score with β<1.** Since providing a loan to a potential defaulter might result in extreme losses to the business.

```
X = loanDF.drop('status', axis=1)
y = loanDF.status
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

2.1. Logistic Regression

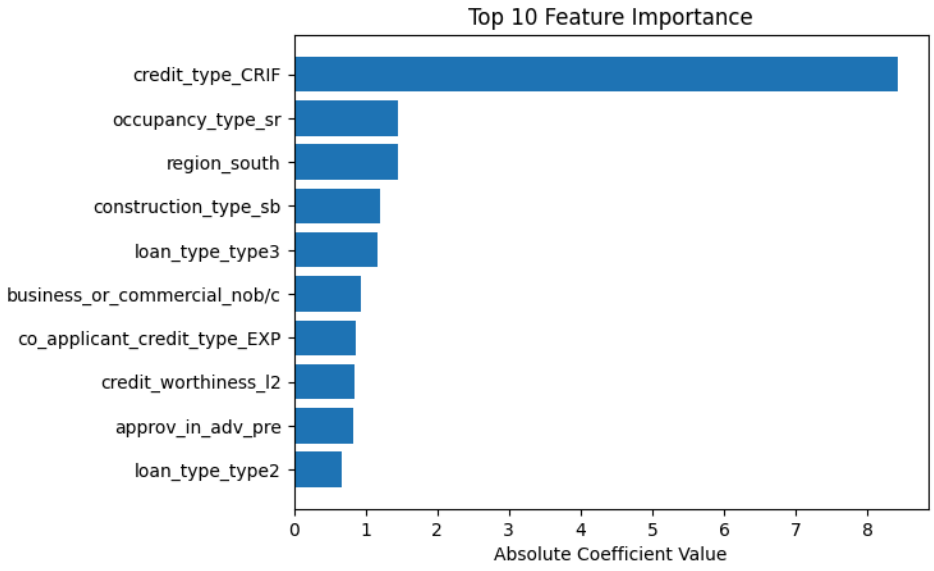
We employed a logistic regression model and trained the model on the training data. Since it is fast and simple, it can be considered as a baseline for evaluating other complex models. After training, we made predictions on the testing dataset using the trained logistic regression model. Then, we printed the classification report that includes precision, recall, f1-score, and accuracy.

```
lr_model = LogisticRegression(solver='lbfgs', max_iter=1000)
```

```
# Train the model
lr_model.fit(X_train, y_train)

# Predict the labels for the test set
y_pred_lr = lr_model.predict(X_test)

print(classification_report(y_test, y_pred_lr))
```

```
              precision    recall  f1-score   support

         0.0       0.72      0.89      0.80      6867
         1.0       0.86      0.67      0.75      7051

    accuracy                           0.77     13918
   macro avg       0.79      0.78      0.77     13918
weighted avg       0.79      0.77      0.77     13918
```

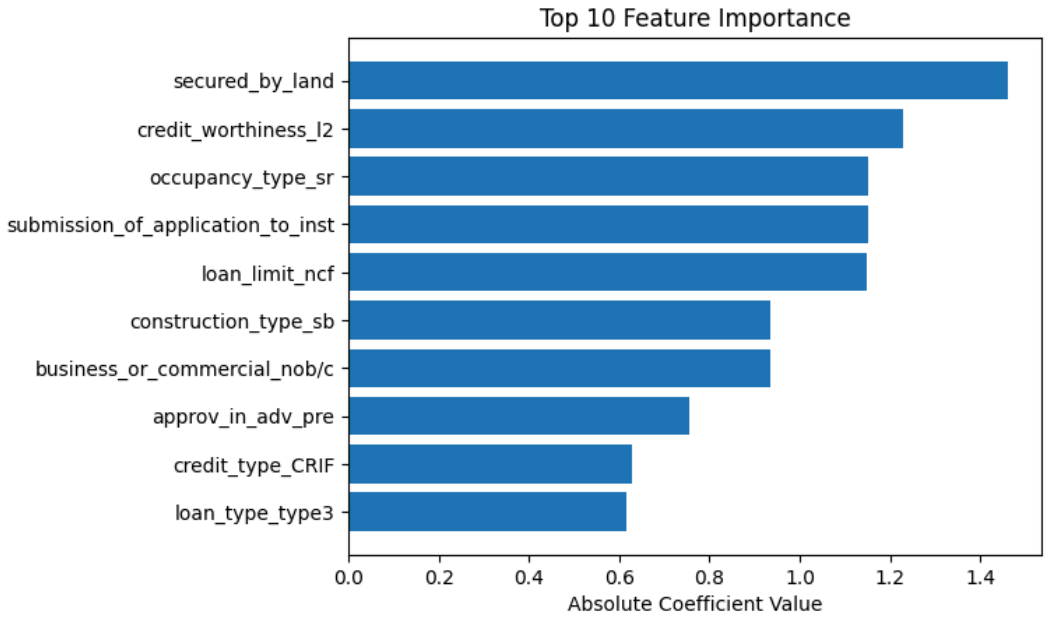
Then, we printed the top 10 feature importance in our logistic regression model. According to the results, ‘credit_type_CRIF’ was the most important feature. However, it is the company that gave the credit report of a customer.



To further evaluate the importance of that feature, we re-trained our model without the ‘credit_type_CRIF’ feature and made predictions on the testing dataset using the trained logistic regression model. Then, we printed the classification report and the top 10 feature importance in our re-trained logistic regression model.

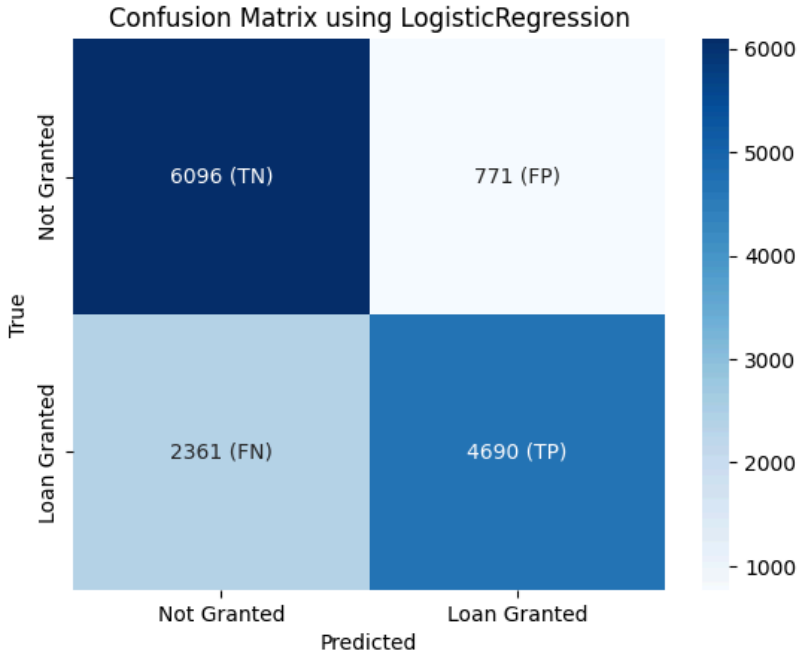
```
new_loan_df = loanDF.drop([col for col in loanDF.columns if col.startswith('credit_type')], axis=1)
X_ = new_loan_df.drop('status', axis=1)
y_ = new_loan_df.status
X_train_, X_test_, y_train_, y_test_ = train_test_split(X_, y_, test_size=0.2, random_state=42)

lr_model_ = LogisticRegression(solver='lbfgs', max_iter=2000)

# Train the model
lr_model_.fit(X_train_, y_train_)

# Predict the labels for the test set
y_pred_lr_ = lr_model_.predict(X_test_)

print(classification_report(y_test_, y_pred_lr_))
```

```
              precision    recall  f1-score   support

         0.0       0.66      0.73      0.69      6867
         1.0       0.71      0.64      0.67      7051

    accuracy                           0.68     13918
   macro avg       0.68      0.68      0.68     13918
weighted avg       0.69      0.68      0.68     13918
```

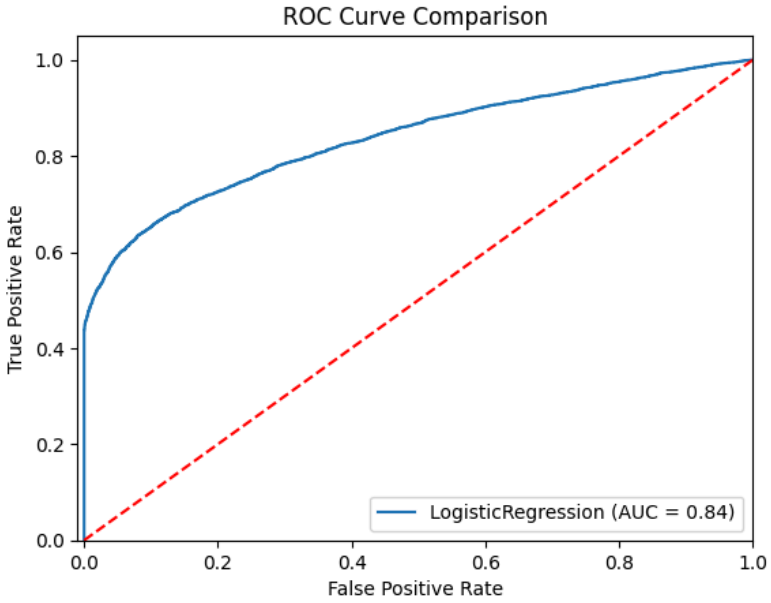


The new model trained without credit_type information achieved a low score. Indicating that the credit_type column is in fact helping the model. Moreover, the features that have high importance like secured_by_land, credit_worthiness, and occupancy_type are good general indicators even in the real world. So, the previous model generalizes better. In addition, the model has fewer False Positives (FP), indicating that it will help us avoid potential loan defaulters.

```python
def plot_confusion_matrix(X_test, y_test, model):
    y_pred = model.predict(X_test)
    # Compute confusion matrix
    cm = confusion_matrix(y_test, y_pred)

    # annotations to include values and corresponding labels
    labels = [["{} (TN)".format(cm[0, 0]), "{} (FP)".format(cm[0, 1])],
            ["{} (FN)".format(cm[1, 0]), "{} (TP)".format(cm[1, 1])]]

    sns.heatmap(cm, annot=labels, fmt='', cmap='Blues',
                xticklabels=['Not Granted', 'Loan Granted'],
                yticklabels=['Not Granted', 'Loan Granted'])

    plt.xlabel('Predicted')
    plt.ylabel('True')
    plt.title(f'Confusion Matrix using {type(model).__name__}')
    plt.savefig(f'../plots-phase2/Confusion Matrix using {type(model).__name__}.png')
    plt.show()


def ROCcurve(X_test, y_test, models, plot_name=None):
    # plt.figure(figsize=(10, 8))
    if not isinstance(models, list):
        models = [models]
    for model in models:
        fpr, tpr, thresholds = roc_curve(y_test, model.predict_proba(X_test)[:,1])
        plt.plot(fpr, tpr, label=f'{type(model).__name__} (AUC = {auc(fpr, tpr):.2f})'

    plt.plot([0, 1], [0, 1], linestyle='--', color='r')

    plt.xlim([-0.01, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC Curve Comparison')
    plt.legend(loc="lower right")
    if plot_name is None:
        plot_name = type(model).__name__
    plt.savefig(f'../plots-phase2/ROC Curve Comparison {plot_name}.png')

    plt.show()
```



We utilized the confusion matrix and ROC curve to visualize the results and analyze the performance of our logistic regression model on the test set. This model achieved an AUC of 0.84, which means the model generalizes well and balances both FPR and TPR.



2.2. KNN

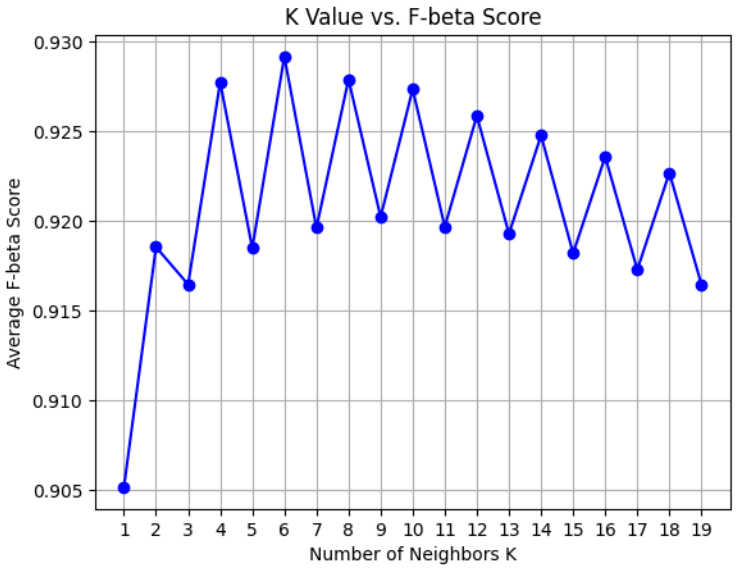
Since loans with similar characteristics might have similar properties. We used K-Neareset Neighbours trained on the training data with a k value of 3. After training, we made predictions on the testing dataset using the trained KNN model. Then, we printed the classification report that includes precision, recall, f1-score, and accuracy.

```
# Train KNN model
knn = KNeighborsClassifier(n_neighbors=3)
```

```
knn.fit(X_train, y_train)
y_pred_knn = knn.predict(X_test)
print(classification_report(y_test, y_pred_knn))
```

```
              precision    recall  f1-score   support

         0.0       0.96      0.88      0.91      6867
         1.0       0.89      0.96      0.92      7051

    accuracy                           0.92     13918
   macro avg       0.92      0.92      0.92     13918
weighted avg       0.92      0.92      0.92     13918
```

We performed cross-validation to find the best-performing k value using ***F-*** β ***Score***. According to the results, the optimal value of k is 6, as shown below.



With the optimal k value, we trained the model, made predictions on the testing dataset, and printed the classification report.

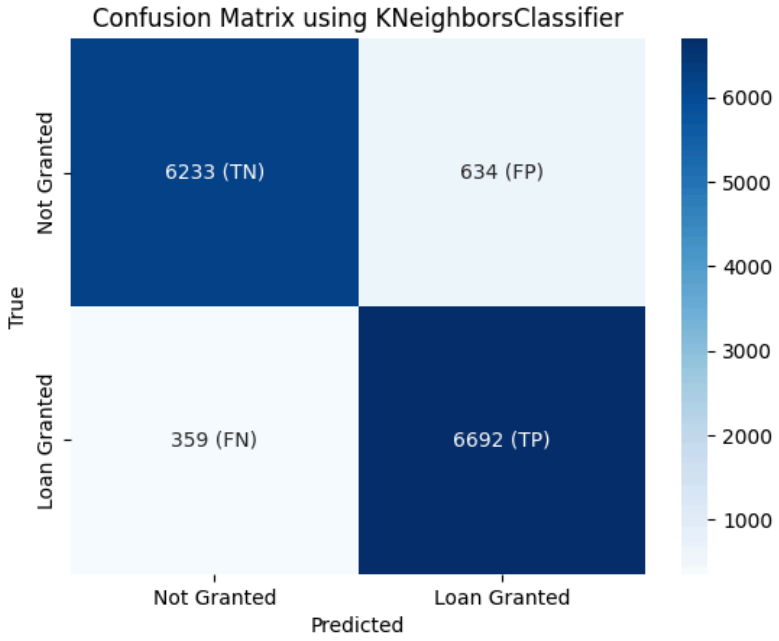
```
avg_scores = []

for k in tqdm(range(1,20)):
    _knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(_knn, X, y, cv=10, scoring=fbeta_scorer, n_jobs=-1)  # 10-fold cross-validation
    avg_scores.append(np.mean(scores))

# Find the k with the highest average score
optimal_k = avg_scores.index(max(avg_scores))+1
print("Optimal value of k:", optimal_k)
```

```
100% 19/19 [02:22<00:00, 7.23s/it]
Optimal value of k: 6
```

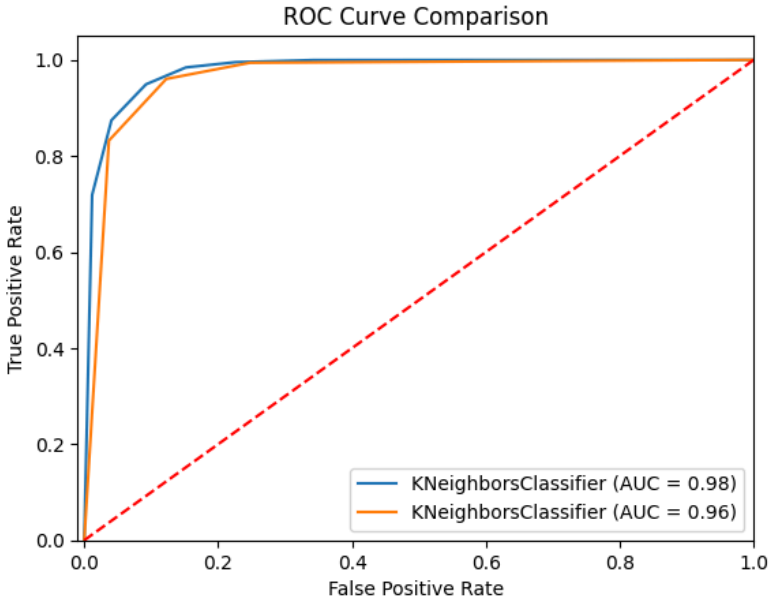
```
# Train KNN model
knn_best = KNeighborsClassifier(n_neighbors=optimal_k)
knn_best.fit(X_train, y_train)
y_pred_knn = knn_best.predict(X_test)
print(classification_report(y_test, y_pred_knn))
```

```
              precision    recall  f1-score   support

         0.0       0.95      0.91      0.93      6867
         1.0       0.91      0.95      0.93      7051

    accuracy                           0.93     13918
   macro avg       0.93      0.93      0.93     13918
weighted avg       0.93      0.93      0.93     13918
```

However, the increase in **the f1-score** is not substantial. We analyzed the confusion matrix and ROC curve to visualize the results and analyze the performance of our KNN model on the test set.



Notably, KNN with K=6 performs better than Logistic regression with very **less FP** and **FN.** To understand the actual improvement between k=3 and 6, we made a ROC curve to find the AUC(Area Under ROC), as below. It can be inferred that the AUC=0.98 is close to 1 when k = 6 and the AUC=0.96 with k=3.



2.3. Naive Bayes

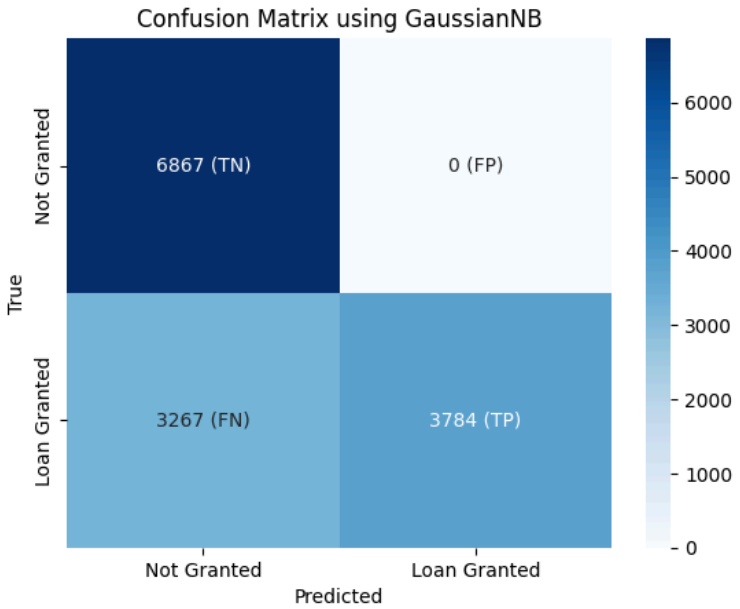
We chose Naive Bayes as it serves as a good baseline model to compare with more complex algorithms. It's fast to train and can handle large datasets efficiently. However, it relies on the assumption that the input features are independent. So, we trained the model on the training data. After training, we made predictions on the testing dataset using the trained Naive Bayes model. Finally, we printed the classification report, which includes precision, recall, F1-score, and accuracy, to evaluate the model's performance.

```
nb_model = GaussianNB()
```

```
# Train the model
nb_model.fit(X_train, y_train)

# Predict the labels for the test set
y_pred_nb = nb_model.predict(X_test)
print(classification_report(y_test, y_pred_nb))
```

```
              precision    recall  f1-score   support

         0.0       0.68      1.00      0.81      6867
         1.0       1.00      0.54      0.70      7051

    accuracy                           0.77     13918
   macro avg       0.84      0.77      0.75     13918
weighted avg       0.84      0.77      0.75     13918
```

We utilized the confusion matrix to visualize the results and analyze the performance of our Naive Bayes model on the test set.



Since, $Precision = \frac{TP}{TP+FP}$, it achieved a perfect precision of 1. This means there are no instances where it incorrectly predicts that a loan would be granted. However, the overall F1 score and recall are lower, indicating that the input features in the dataset are not independent, which is typically the case in real-world datasets.

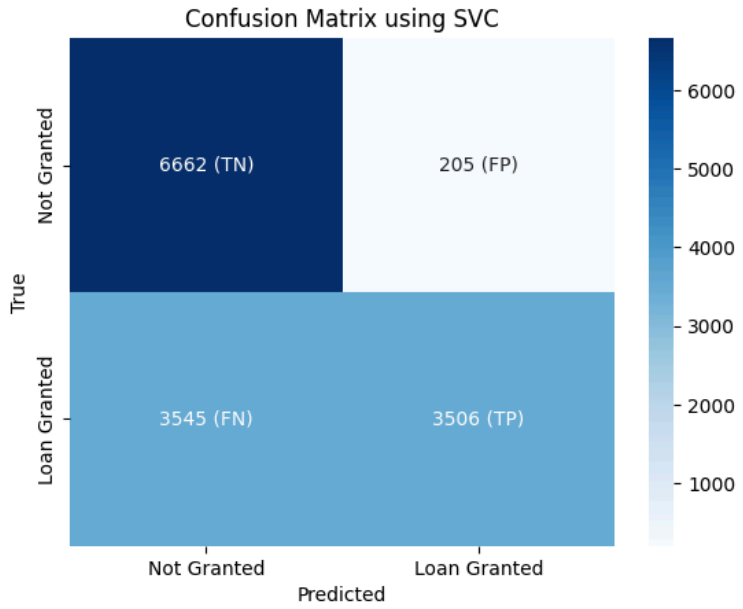
2.4. Support Vector Machine(SVM)

We implemented a Support Vector Machine (SVM) model and trained it on our training dataset. Once the model was trained, we used it to make predictions on our test dataset. In our exploratory data analysis (EDA) in phase 1, we observed that several features are not linearly related to each other, indicating the presence of non-linear relationships. Therefore, employing an SVM for modeling, which can handle such non-linearities through kernel functions, could potentially lead to better generalization.

```
print(classification_report(y_test, y_pred_svm))
```
✓ 0.0s

```
              precision    recall  f1-score   support

         0.0       0.65      0.97      0.78      6867
         1.0       0.94      0.50      0.65      7051

    accuracy                           0.73     13918
   macro avg       0.80      0.73      0.72     13918
weighted avg       0.80      0.73      0.72     13918
```

Overall, the SVC achieved a decent f1-score of 73% but to further analyze the performance. We utilized the confusion matrix to visualize the results and analyze the performance of our SVM model on the test set. Similar to all the other models SVM, also maintained a higher precision sacrificing recall. In addition, we printed the parameters after training.




```
# Print the parameters after training
# print("Support vectors:\n", svm_model.support_vectors_)
print("Indices of support vectors:", svm_model.support_)
print("Number of support vectors for each class:", svm_model.n_support_)
print("Coefficients of the support vector in the decision function:", svm_model.dual_coef_)
if svm_model.kernel == 'linear':
    print("Weights assigned to the features (coef_):", svm_model.coef_)
print("Constants in decision function (intercept_):", svm_model.intercept_)
```

✓ 0.0s

```
Indices of support vectors: [    0     4    21 ... 55660 55662 55663]
Number of support vectors for each class: [20428 20412]
Coefficients of the support vector in the decision function: [[-1. -1. -1. ...  1.  1.  1.]]
Constants in decision function (intercept_): [-3.19271449]
```

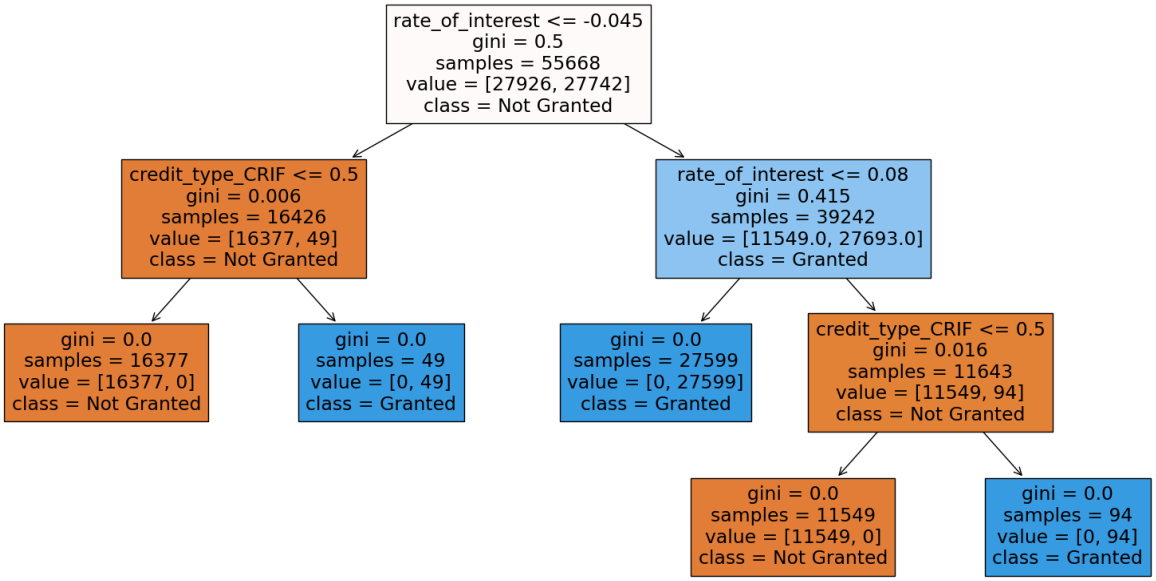
2.5. Decision Tree

We first initialized the Decision Tree classifier and trained the model. After training, we made predictions on the testing dataset using the trained Decision Tree model. The below image is our generated decision tree.

```
# Initialize Decision Tree classifier
dt_model = DecisionTreeClassifier()

# Train the model
dt_model.fit(X_train, y_train)
```

▾ DecisionTreeClassifier

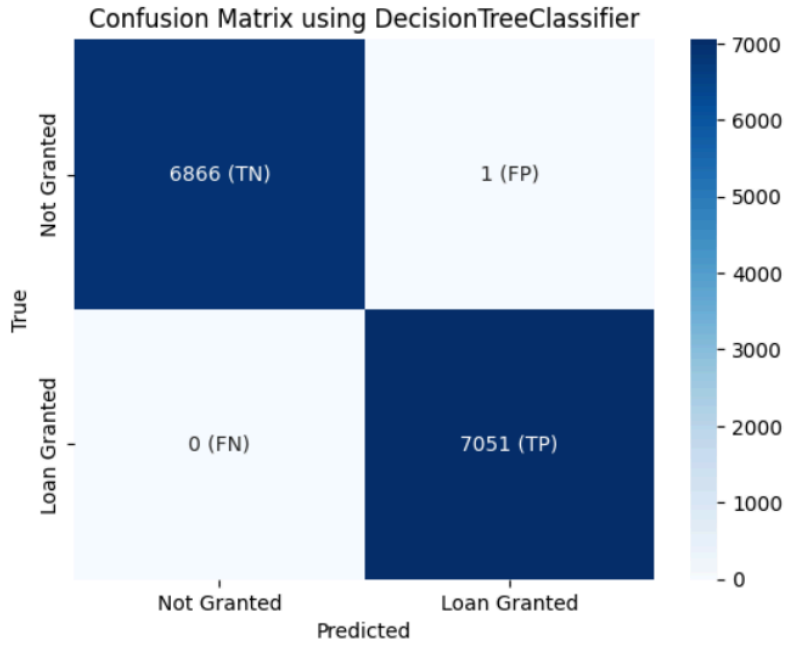
DecisionTreeClassifier()



We utilized the confusion matrix to visualize the results and analyze the performance of our Decision Tree model on the test set.

```
# Predict the labels for the test set
y_pred_dt = dt_model.predict(X_test)
```

```
plot_confusion_matrix(X_test, y_test, dt_model)
```



2.6. RandomForest

Tree-based models like Decision trees achieved the highest accuracy. We tried to use more robust and efficient algorithms like Random Forest and trained the model on the training data.

```
rf_model = RandomForestClassifier(n_estimators=3, random_state=1)

# Train the model
rf_model.fit(X_train, y_train)
```

✓ 0.0s

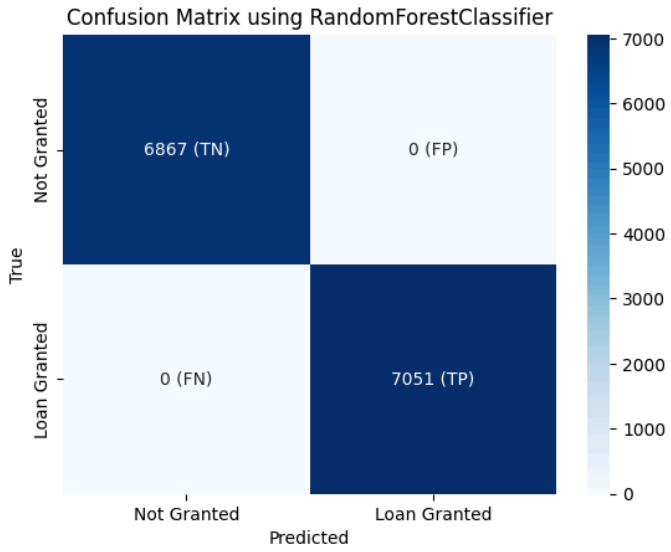
```
RandomForestClassifier(n_estimators=3, random_state=1)
```

After training, we made predictions on the testing dataset using the trained Random Forest model. Then, we printed the classification report that includes precision, recall, f1-score, and accuracy.

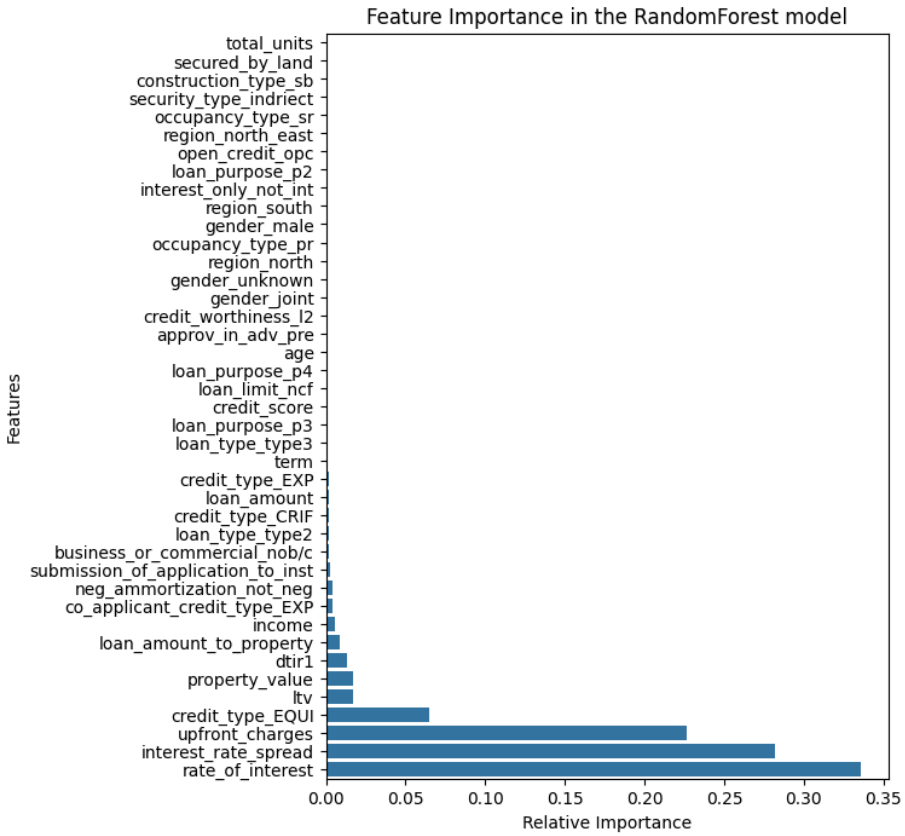
```
# Predict the labels for the test set
y_pred_rf = rf_model.predict(X_test)
print(classification_report(y_test, y_pred_rf))
```

```
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00      6867
         1.0       1.00      1.00      1.00      7051

    accuracy                           1.00     13918
   macro avg       1.00      1.00      1.00     13918
weighted avg       1.00      1.00      1.00     13918
```

We utilized the confusion matrix to visualize the results and analyze the performance of our Random Forest model on the test set. Unlike, decision tree random forest achieved a perfect score on test data.



In addition, we displayed the feature importances in our Random Forest model using a horizontal bar plot.



From the above image, both decision tree and Random Forest utilized rate_of_interest and credit_type like other machine learning models.

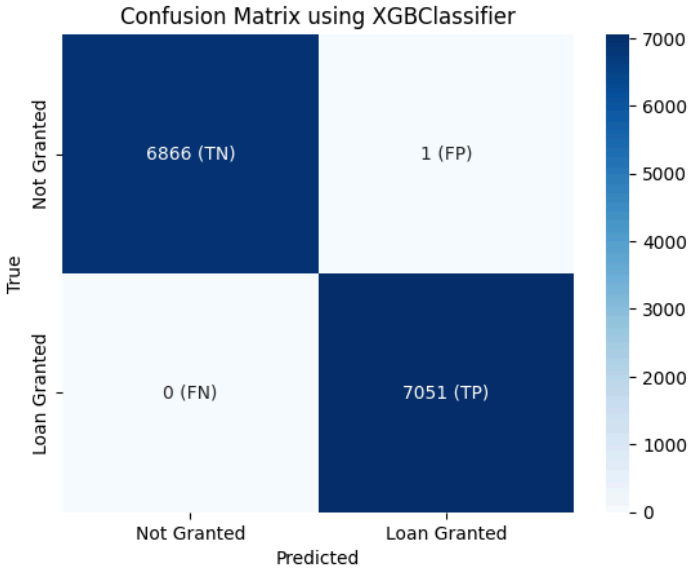
2.7. XGBoost

Finally, we tried using XGBoost which is known for performance, efficiency, and versatility. We employed a XGBoost model and trained the model on the training data.

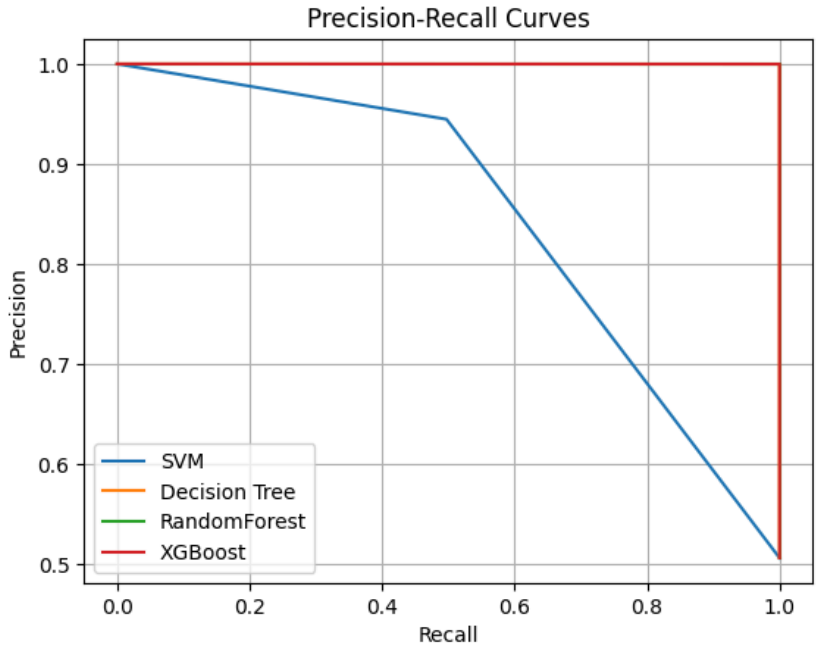
```
# Initialize XGBoost classifier
xgb_model = xgb.XGBClassifier()

# Train the model
xgb_model.fit(X_train, y_train)
```

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=None, n_jobs=None,
              num_parallel_tree=None, random_state=None, ...)
```

After training, we made predictions on the testing dataset using the trained XGBoost model. We utilized the confusion matrix to visualize the results and analyze the performance of our XGBoost model on the test set. In addition, we plot precision-recall curves for SVM, Decision Tree, and XGBoost.



The above confusion matrix indicates similar results of tree based algorithms like Decision tree and Random Forest. The same can be inferred from the above precision-recall curve.

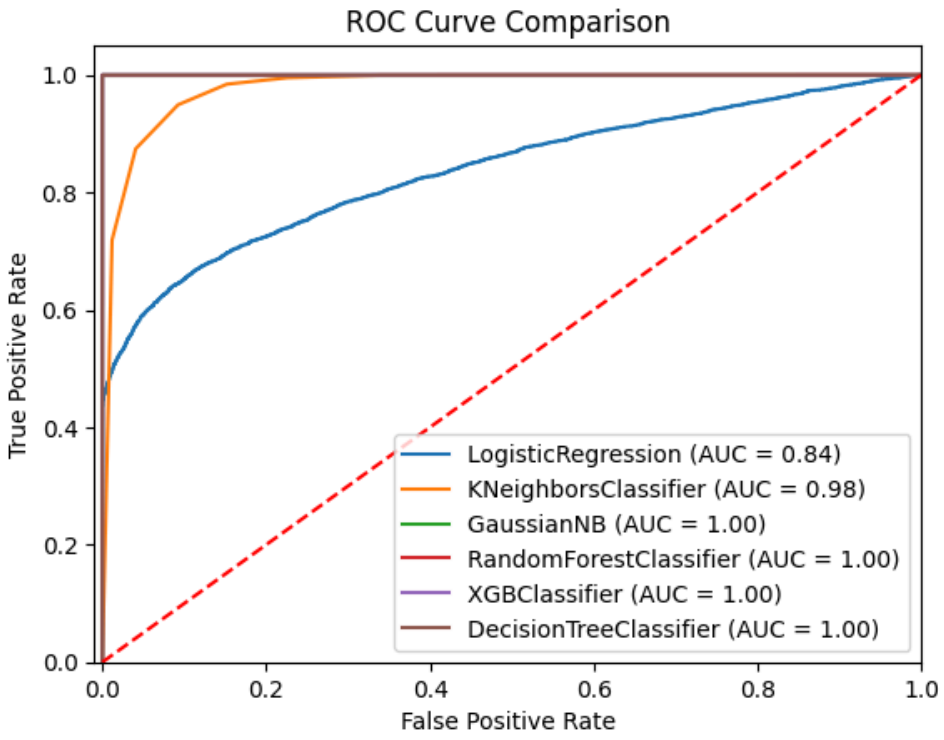


2.8. Select the Best Model using Cross Validation

Since we have multiple models that perform better. In order to select the best model, we first defined the models: Logistic Regression, KNN, Random Forest, Naive Bayes, SVM, XGBoost, and Decision Tree. Then, we created pipelines for each model to perform cross-validation to see which model generalizes better. Since cross-validation ensures that the model is generalizing well for the entire dataset irrespective of the order. Using it to select the best model is a better approach. Here, we used **cv=5.**

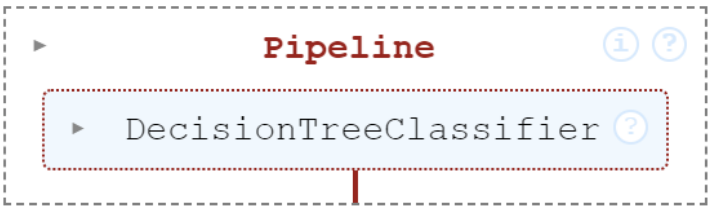
```python
# Define the models
models = {
    "Logistic Regression": lr_model_,
    "KNN": knn_best,
    "Naive Bayes": nb_model,
    "SVM": svm_model,
    "Random Forest": rf_model,
    "XGBoost": xgb_model,
    "Decision Tree": dt_model,
}

# Create pipelines for each model
pipelines = {name: Pipeline([ ('model', model)]) for name, model in models.items()}
```

Each model was evaluated using cross-validation, and its performance was printed.



Given, the performance, simplicity, and interpretability, we can choose Decision Tree as our best model.

```
best_model
✓ 0.0s
```



3. Explanation and Analysis

3.1. Logistic Regression

In our problem, we classify loan default as default or non-default according to status 0 and 1. Since logistic regression models the relationship between the features and the target variable even if the relationship is nonlinear, it is a good choice for our classification tasks. We applied robust scaling by using 'RobustScaler()' to normalize numerical features in phase 1 and split the data into training and testing sets to prevent overfitting. According to the results of the top 10 feature importance in our logistic regression model, 'credit_type_CRIF' was the most important feature, so we re-trained our model without the 'credit_type_CRIF' feature since it is just the company that gave the credit report of a customer. The accuracy of the model without 'credit_type_CRIF' is 77%, which is the same accuracy as the model with the 'credit_type_CRIF'. According to the results of the top 10 feature importance in our logistic regression model without 'credit_type_CRIF', the top 10 features that have high importance such as 'secured_by_land', 'credit_worthiness', and 'occupancy_typ_sr' are good general indicators. Therefore, this model generalizes better. Furthermore, the new model without 'credit_type_CRIF' has less False Positive Rate(FPR) compared to the model with 'credit_type_CRIF', which indicates that the new model will help us in avoiding potential loan defaulters.

3.2. K-NN

The Knn algorithm is generally robust to outliers when appropriately tuned, which has less impact on the classification task but can be sensitive to outliers for small values of k. It is a good choice for our problem where outliers may represent extreme cases because its robustness can help ensure the reliability of the model's predictions. We applied robust scaling by using 'RobustScaler()' to normalize numerical features in phase 1 and split the data into training and testing sets to prevent overfitting. We performed cross-validation to find the best-performing k value using fbeta_scorer. According to the results, the optimal value of k is 6. With the optimal k value, we trained the model. The accuracy of the model with the optimal k value is 93% whereas the accuracy of the model with a non-optimal k value (k=3) is 92%. Furthermore, the AUC of the ROC curve of the model with the optimal k value is higher than the model with a non-optimal k value(k=3). This indicates that the KNN model with an optimal k value is an effective way to avoid potential loan defaulters.

3.3. Naive Bayes

We chose Naive Bayes as it serves as a good baseline model to compare with more complex algorithms. It's fast to train and can handle large datasets efficiently. However, it relies on the assumption that the input features are independent. We applied robust scaling by using 'RobustScaler()' to normalize numerical features in phase 1 and split the data into training and testing sets to prevent overfitting. According to the results, the accuracy of the model is 77%, and the model achieved a perfect precision of 1, indicating that there are no instances where it incorrectly predicts that a loan would be granted. On the other hand, it achieved 0.54 for recall and 0.7 for the f1 score, indicating that the input features in the dataset are not independent, which is typically the case in real-world datasets.

3.4. SVM

SVMs are often robust to outliers because they focus on maximizing the margin between classes. Furthermore, SVMs are capable of modeling complex non-linear relationships by employing various kernel functions, allowing them to perform well on datasets where the relationship between features is not linear. In our exploratory data analysis (EDA) in phase 1, we observed that several features are not linearly related to each other, indicating the presence of non-linear relationships. Therefore, employing an SVM for modeling, which can handle such non-linearities through kernel functions, could potentially lead to better generalization. We applied robust scaling by using 'RobustScaler()' to normalize numerical features in phase 1 and split the data into training and testing sets to prevent overfitting. Overall, the SVC achieved a decent f1-score of 73% but to further analyze the performance. In addition, it maintained a higher precision sacrificing recall, similar to all the other models. According to the parameters after training, total support vectors of each class are similar in number indicating that the model is not favoring one class over the other.

3.5. Decision Tree

The decision tree is mostly known for its interpretability by allowing clear visualization and understanding of how decisions are made. During the construction of a decision tree, the most informative features are selected for splitting the data. As a result, they do feature selection. Moreover, since they can also model non-linear relationships using a decision tree is perfect for our scenario. We applied robust scaling by using 'RobustScaler()' to normalize numerical features in phase 1 and split the data into

training and testing sets to prevent overfitting. According to our generated decision tree, 'credit_type_CRIF' is also used by the decision tree indicating its importance. Surprisingly, this model only used two features 'rate_of_interest' and 'credit_type_CRIF', and achieved an almost perfect prediction with nearly zero(FP and FN) on the test data. Since a Decision Tree provides a clear and interpretable decision-making process, it helps lenders understand how the model arrives at its predictions.

3.6. Random Forest

In our problem, high accuracy is required because loan default is significantly important, especially for financial institutions. Since the Random Forest algorithm is good for its high accuracy and ability to handle nonlinear relationships, it is good for our classification task. In addition, Random Forest can handle large datasets with a high number of features efficiently, so it is a good choice. Moreover, tree-based models like Decision trees achieved the highest accuracy so we tried to use more robust and efficient algorithms like Random Forest. We applied robust scaling by using 'RobustScaler()' to normalize numerical features in phase 1 and split the data into training and testing sets to prevent overfitting. The accuracy of the model is 100%, and the plot of the feature importance in our Random Forest classifier model shows the relative importance of the following features, with the highest features being 'rate_of_interest', 'interest_rate_spread', and 'upfront_charges'. Since Random Forest can indicate which features are most important in predicting loan defaults, understanding these key factors can help lenders prioritize their assessments. In addition, the accuracy helps lenders gauge the reliability of using the models in their decision-making process.

3.7. XGBoost

Finally, we tried using XGBoost which is known for performance, efficiency, and versatility, which is a good choice for our problem. We applied robust scaling by using 'RobustScaler()' to normalize numerical features in phase 1 and split the data into training and testing sets to prevent overfitting. According to the confusion matrix, it achieved similar results to tree-based algorithms like Decision Tree and Random Forest. The same can be inferred from the above precision-recall curve.

4. References

[1] https://www.kaggle.com/datasets/yasserh/loan-default-dataset

[2] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[3] XGBoost Documentation — xgboost 2.0.3 documentation

[4] sklearn.tree.DecisionTreeClassifier — scikit-learn 1.4.1 documentation