MBA786M: Alert Models in Finance

Group Project:

Yashveer Yadav (211200) - 25% (Ques1)
Ujjawal Dubey (211121) - 25% (Ques3)
Chetanya Bhan (210283) - 25% (Ques4)
Anudeep Reddy (210165) - 25% (Ques2)

Problem Statement

The task at hand involves predicting whether a customer applying for a loan at a bank will belong to either the "Good" or "Bad" class based on available financial data. A customer classified as "Good" is expected to successfully repay the loan, while a "Bad" classification indicates a likelihood of default. This distinction is crucial for banks when assessing loan applications, as it directly impacts the decision of whether to approve a loan, ensuring the bank minimizes its risk and maximizes the likelihood of loan repayment. The challenge is to develop a predictive model that accurately classifies customers, aiding the bank in making informed lending decisions.

Approach

In this analysis, we will train various machine learning models—Logistic Regression, K-Nearest Neighbors (KNN), Tree-based models, and Support Vector Machines (SVM)—to predict the classification of customers based on their financial data. Each model will be evaluated for its ability to accurately categorize customers into "Good" or "Bad" classes. The performance of these models will be compared using the Confusion Matrix as a key evaluation metric, enabling us to analyze and contrast the models' predictive power in terms of true positive rate, false positive rate, precision, and overall accuracy. Through this comparative approach, we aim to identify the most reliable model for making loan approval decisions.

Project Tasks

Question1

- Part-A:

- # Problem Statement:

  Fit a logistic regression model on the dataset. Choose a probability of default threshold of 20%, 35%, and 50%, to assign an observation to the Bad class. Compute a confusion matrix for each of the models. How do the True Positive and False Positive rates vary over these models? Which model would you choose?

- # Approach:

  - Applying a logistic regression model directly to categorical outputs like "Good" or "Bad" is not feasible. Therefore, we must encode these classes into numeric values. For this model, I have assigned the value 1 to the "Good" class and 0 to the "Bad" class. This decision is driven by the fact that our dataset is imbalanced, with a higher proportion of observations classified as "Good" compared to "Bad." Through experimentation, it was observed that the model's performance significantly improves when the "Good" class is represented as 1, rather than when the "Bad" class is assigned 1. This encoding choice better aligns with the data's structure, enhancing the model's predictive capabilities.
  - Not all independent variables exhibit a strong correlation with the target variable. Training a model using all independent variables may introduce the risk of overfitting, where the model performs well on the training data but fails to generalize to unseen data. To mitigate this, we initially train the model with all independent variables and then analyze the model's summary to refine it. By leveraging the Null Hypothesis and examining the p-values (Pr), we eliminate variables that are statistically insignificant. Specifically, we retain only those independent variables with p-values below 0.05, ensuring they have a meaningful impact on the model's predictions.
  - The key independent variables identified through this process are: Duration, Amount, InstallmentRatePercentage, ForeignWorker, CheckingAccountStatus.lt.0, CheckingAccountStatus.0.to.200, CreditHistory.NoCredit.AllPaid, CreditHistory.ThisBank.AllPaid, CreditHistory.PaidDuly, SavingsAccountBonds.lt.100, OtherDebtorsGuarantors.None, OtherDebtorsGuarantors.CoApplicant, and OtherInstallmentPlans.Bank. These variables are deemed crucial for the model's performance.

- # Results:

  1. P(Class = 1) > 0.5 means the class is good. This statement is the same as P(Class = 0) > 0.5 means the class is bad.

     Confusion Matrix

Actual / Predicted	Bad	Good
Bad	134	67

Good	166	633

TP = 134, TN = 633, FP = 67, FN = 166
Accuracy = 0.767
TPR = 0.446666666666667
FPR = 0.0957142857142857
FNR = 0.553333333333333
TNR = 0.904285714285714

2. P(Class = 1) > 0.65 means good class is the same as P(Class = 0) > 0.35 means bad Class.
   Confusion Matrix

Actual / Predicted	Bad	Good
Bad	200	161
Good	100	539

   TP = 200, TN = 539, FP = 161, FN = 100
   Accuracy = 0.739
   TPR = 0.666666666666667
   FPR = 0.23
   FNR = 0.333333333333333
   TNR = 0.77

3. P(Class = 1) > 0.8 means good class is the same as P(class = 0) > 0.2 means bad class.
   Confusion Matrix

Actual / Predicted	Bad	Good
Bad	255	313
Good	45	387

   TP = 255, TN = 387, FP = 313, FN = 45
   Accuracy = 0.642
   TPR = 0.85
   FPR = 0.447142857142857
   FNR = 0.15
   TNR = 0.552857142857143

- Observation:

TPR (True Positive Rate): Increases as the threshold increases.
From **0.4467** at P(Class=1)>0.5 to **0.85** at P(Class=1)>0.8.
This indicates that higher thresholds result in more true positives (correct predictions of Good Class).

FPR (False Positive Rate): Also increases with higher thresholds.
From **0.0957** at P(Class=1)>0.5 to **0.4471** at P(Class=1)>0.8.
This shows that as the threshold increases, the model misclassifies more Bad Class instances as Good Class.

- ## Model Choice:

Since in a banking setting TPR are very important. One Bad Class assigned as good means the bank will lose its money. So in this FPR vs TPR trade off. **I will prefer high TPR thus will use as P(Class = 0) > 0.2 means Bad class as the threshold for predicting bad class in the model.**

- # Part-B

- Problem Statement:

Divide the dataset into training (70%) and test (30%) sets and repeat the above question and report the performance of these models on the test set.

- Result

- P(Class = 0) > 0.5 means Bad Class

Confusion Matrix

Actual / Predicted	Bad	Good
Bad	40	26
Good	52	182

TP = 40, TN = 182, FP = 26, FN = 52
Accuracy = `0.74`
TPR = `0.434782608695652`
FPR = `0.125`
FNR = `0.565217391304348`
TNR = `0.875`

- P(Class = 0) > 0.35 means Bad Class

Confusion Matrix

Predicted \ Actual	Bad	Good
Bad	56	59
Good	36	149

TP = 56, TN = 149, FP = 59, FN = 36
Accuracy = 0.683333333333333
TPR = 0.608695652173913
FPR = 0.283653846153846
FNR = 0.391304347826087
TNR = 0.716346153846154

- P(Class = 0) > 0.2 means bad Class

Confusion Matrix

Predicted \ Actual	Bad	Good
Bad	69	90
Good	23	118

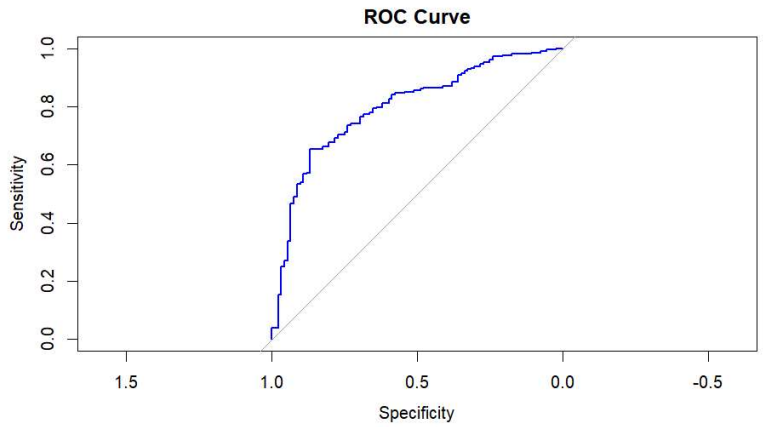
TP = 69, TN = 118, FP = 90, FN = 23
Accuracy = 0.623333333333333
TPR =0.75
FPR = 0.432692307692308
FNR = 0.25
TNR = 0.567307692307692

- PartC:

-Problem statement :

Plot the ROC for a logistic model on a graph and compute the AUC. Explain the information conveyed by the ROC and the AUC metrics.

- Results



The AUC value is `0.796404682274247`

The ROC curve shows the trade-off between sensitivity (true positive rate) and specificity for different thresholds in a logistic regression model. The diagonal line represents random guessing, while the curve indicates how well the model performs. Since the curve lies above the blue line means the model performs well then random guessing. The AUC (Area Under the Curve) of 0.796 means the model has good classification ability, correctly distinguishing between "Good Class" and "Bad Class" about 79.6% of the time. AUC closer to 1 indicates better performance.

Question2

Decision Tree- Full

```
Confusion Matrix and Statistics

          Reference
Prediction  Bad Good
       Bad  147   38
       Good 153 662

               Accuracy : 0.809
                 95% CI : (0.7832, 0.8329)
    No Information Rate : 0.7
    P-Value [Acc > NIR] : 2.857e-15

                  Kappa : 0.4893

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.4900
            Specificity : 0.9457
```

```
         Pos Pred Value : 0.7946
         Neg Pred Value : 0.8123
             Prevalence : 0.3000
         Detection Rate : 0.1470
   Detection Prevalence : 0.1850
      Balanced Accuracy : 0.7179

       'Positive' Class : Bad
```

Bagging- Full

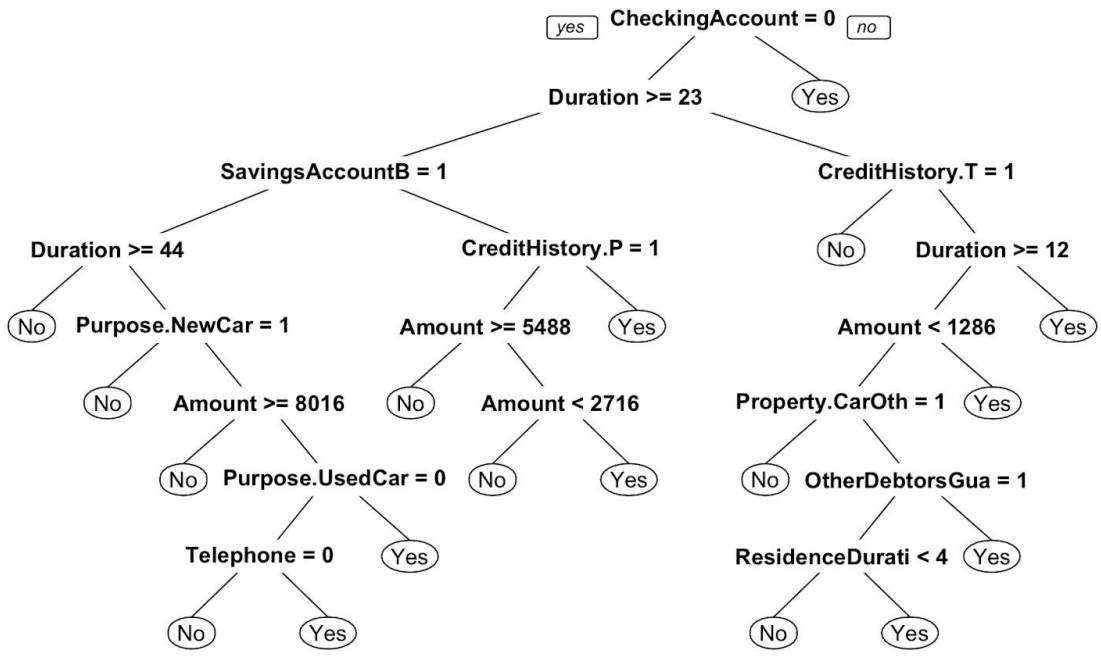
```
Confusion Matrix and Statistics

          Reference
Prediction  Bad Good
       Bad  300    0
       Good   0  700

               Accuracy : 1
                 95% CI : (0.9963, 1)
    No Information Rate : 0.7
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 1

 Mcnemar's Test P-Value : NA

            Sensitivity : 1.0
            Specificity : 1.0
         Pos Pred Value : 1.0
         Neg Pred Value : 1.0
             Prevalence : 0.3
         Detection Rate : 0.3
   Detection Prevalence : 0.3
      Balanced Accuracy : 1.0

       'Positive' Class : Bad
```

RANDOM FOREST-FULL

```
Confusion Matrix and Statistics

          Reference
Prediction  Bad Good
       Bad  300    0
       Good   0  700

               Accuracy : 1
```

```
                95% CI : (0.9963, 1)
    No Information Rate : 0.7
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 1

 Mcnemar's Test P-Value : NA

            Sensitivity : 1.0
            Specificity : 1.0
         Pos Pred Value : 1.0
         Neg Pred Value : 1.0
             Prevalence : 0.3
       Detection Rate : 0.3
   Detection Prevalence : 0.3
      Balanced Accuracy : 1.0

       'Positive' Class : Bad
```

a)



Decision Tree, when trained on full data.

Comparison of Model Performance

Based on the provided confusion matrices, we can observe the following:

- **Decision Tree:** Achieves an accuracy of 0.809, with a sensitivity of 0.49 and a specificity of 0.9457. While the accuracy is reasonable, the sensitivity is relatively low, indicating that the model might struggle to correctly predict the "No" class.
- **Bagging:** Demonstrates perfect accuracy (1.0) with perfect sensitivity and specificity. This suggests that the bagging ensemble has effectively overfitted to the data.
- **Random Forest:** Also achieves perfect accuracy (1.0), indicating that the random forest ensemble has overfitted and learned a strong predictive model which has high variance.

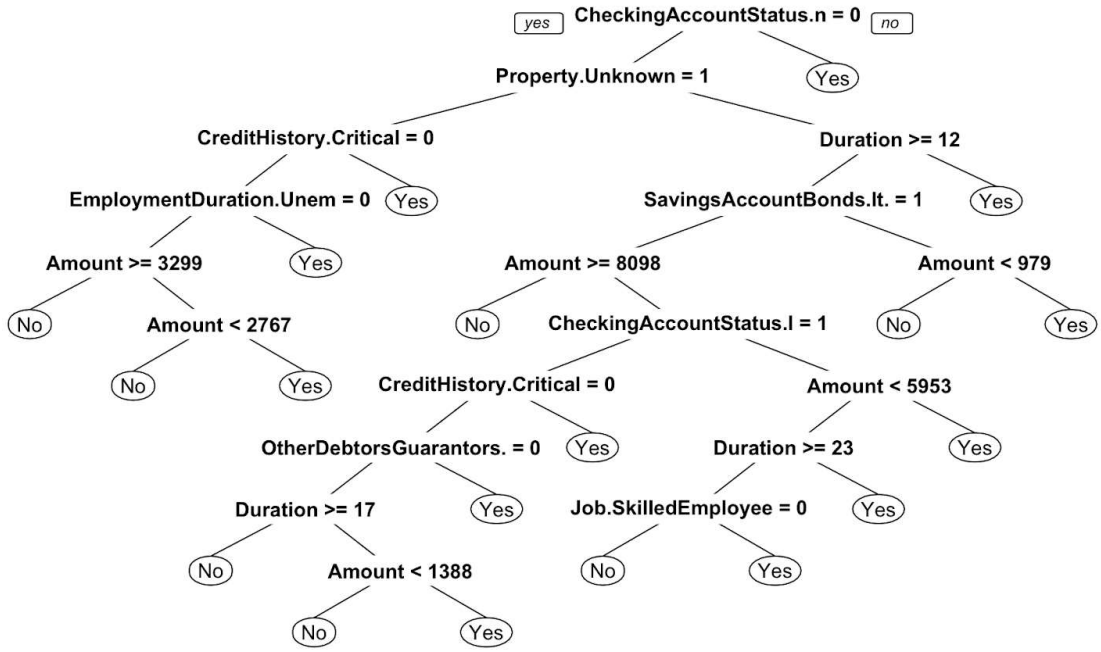
Overfitting Concerns and Rationale

Yes, we might be overestimating the performance of these models by fitting them on the entire dataset.

Here's why:

1. **Lack of Independent Test Set:** By training and evaluating the models on the same dataset, we risk overfitting,where the models learn to memorize the training data rather than generalize to unseen data.
2. **Perfect Accuracy:** The perfect accuracy achieved by bagging and random forest on the training data raises concerns about overfitting. While these models are designed to reduce overfitting, they might still be memorizing patterns in the training data.

b)



Decision Tree- Train test split

```
Confusion Matrix and Statistics

          Reference
Prediction  Bad Good
       Bad   32  20
       Good  58 190

               Accuracy : 0.74
                 95% CI : (0.6865, 0.7887)
    No Information Rate : 0.7
    P-Value [Acc > NIR] : 0.07228

                  Kappa : 0.296

 Mcnemar's Test P-Value : 2.797e-05

            Sensitivity : 0.3556
            Specificity : 0.9048
         Pos Pred Value : 0.6154
         Neg Pred Value : 0.7661
             Prevalence : 0.3000
       Detection Rate : 0.1067
   Detection Prevalence : 0.1733
      Balanced Accuracy : 0.6302

  'Positive' Class : Bad
```

Bagging- train test split

```
Confusion Matrix and Statistics

          Reference
Prediction  Bad Good
       Bad   42  29
       Good  48 181

               Accuracy : 0.7433
                 95% CI : (0.69, 0.7918)
    No Information Rate : 0.7
    P-Value [Acc > NIR] : 0.05608

                  Kappa : 0.3497

 Mcnemar's Test P-Value : 0.04024

            Sensitivity : 0.4667
            Specificity : 0.8619
         Pos Pred Value : 0.5915
         Neg Pred Value : 0.7904
            Prevalence : 0.3000
         Detection Rate : 0.1400
   Detection Prevalence : 0.2367
      Balanced Accuracy : 0.6643

       'Positive' Class : Bad
```

RANDOM FOREST- TRAIN TEST SPLIT

```
Confusion Matrix and Statistics

          Reference
Prediction  Bad Good
       Bad   37  16
       Good  53 194

               Accuracy : 0.77
                 95% CI : (0.7182, 0.8164)
    No Information Rate : 0.7
    P-Value [Acc > NIR] : 0.004167

                  Kappa : 0.3795
```

```
Mcnemar's Test P-Value : 1.465e-05

            Sensitivity : 0.4111
            Specificity : 0.9238
         Pos Pred Value : 0.6981
         Neg Pred Value : 0.7854
            Prevalence : 0.3000
         Detection Rate : 0.1233
   Detection Prevalence : 0.1767
      Balanced Accuracy : 0.6675

       'Positive' Class : Bad
```

Even though all the models perform closely, we can find that we get the best accuracy using random forest with 77% accuracy.
In banking, accurately predicting "bad customers" (those likely to default on loans or engage in fraudulent activities) is crucial for risk management and financial stability. Given this context, a model with **high sensitivity** is particularly important.

Sensitivity: The proportion of actual "bad customers" that are correctly identified by the model.

In banking, it's essential to minimise the number of "bad customers" who are not flagged by the model. False negatives can lead to significant financial losses.

Based on the provided results:

Bagging has the highest sensitivity (0.4667).

Random Forest has a slightly lower sensitivity (0.4111).

Decision Tree has the lowest sensitivity (0.3556).

Given the importance of identifying "bad customers" in banking, **Bagging** appears to be the most suitable model. It has the highest sensitivity, which means it is more likely to correctly identify potential risks.

But Bagging also has high False Positive Rate compared to Random forest model, which means that the bank might reject credit for people who were misclassified as bad even though they are good.

If Accuracy is the only metric to be looked at, then Random Forest performs the best with 77% accuracy, Proving that it indeed is a robust method to unseen data with low variance than that of Bagging.

Looking at the Feature Importance, one can say that bagging is heavily dependent on Amount with over 50% MeanDecreaseGini, which is over 30% in Random Forest. Considering this, one can say that Random Forest Model is better model for out of sample data, as there might be a possibility that the training data has patterns related to amount which the model might have learnt well, leading to higher variance. This is the reason why random forest is used, where few predictors are not used in all the trees, leading to better overall performance in out of sample data.

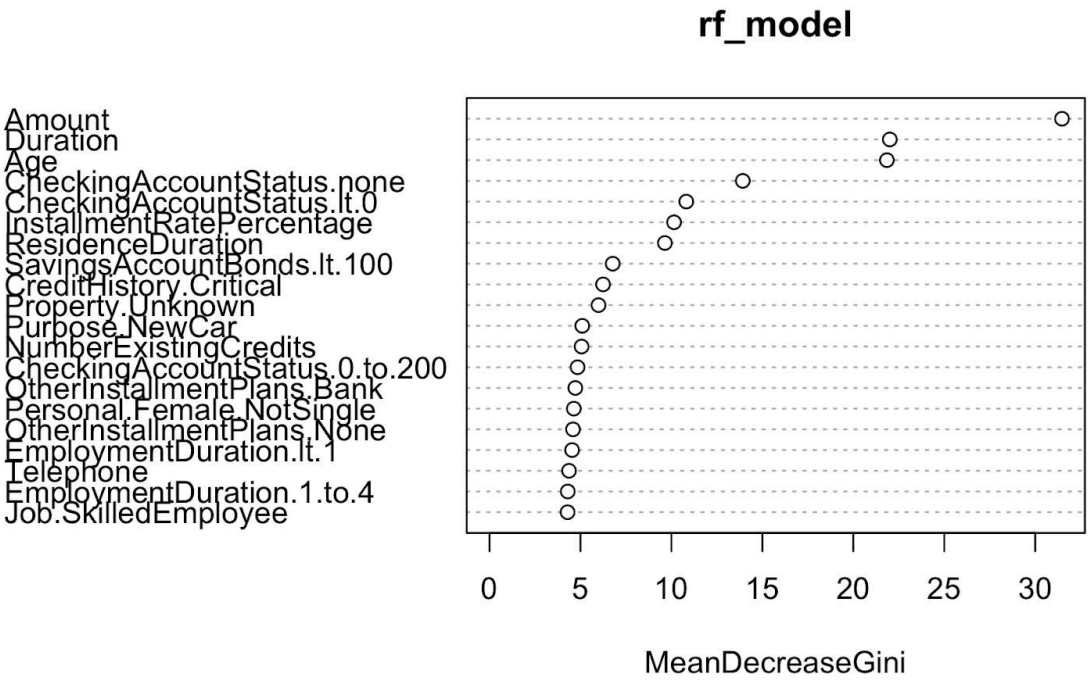
c)

Top 25 features importances are shown for visual appeal

For Random Forest,

	overall	**names**
	31.4717907	Amount
	22.0077559	Duration
	21.8467087	Age
	13.9245087	CheckingAccountStatus.none
	10.8173553	CheckingAccountStatus.lt.0
	10.1440231	InstallmentRatePercentage
	9.6418584	ResidenceDuration
	6.7744559	SavingsAccountBonds.lt.100
	6.2423453	CreditHistory.Critical
	5.9928309	Property.Unknown
	5.0991768	Purpose.NewCar
	5.0625306	NumberExistingCredits
	4.8394654	CheckingAccountStatus.0.to.200
	4.7175668	OtherInstallmentPlans.Bank

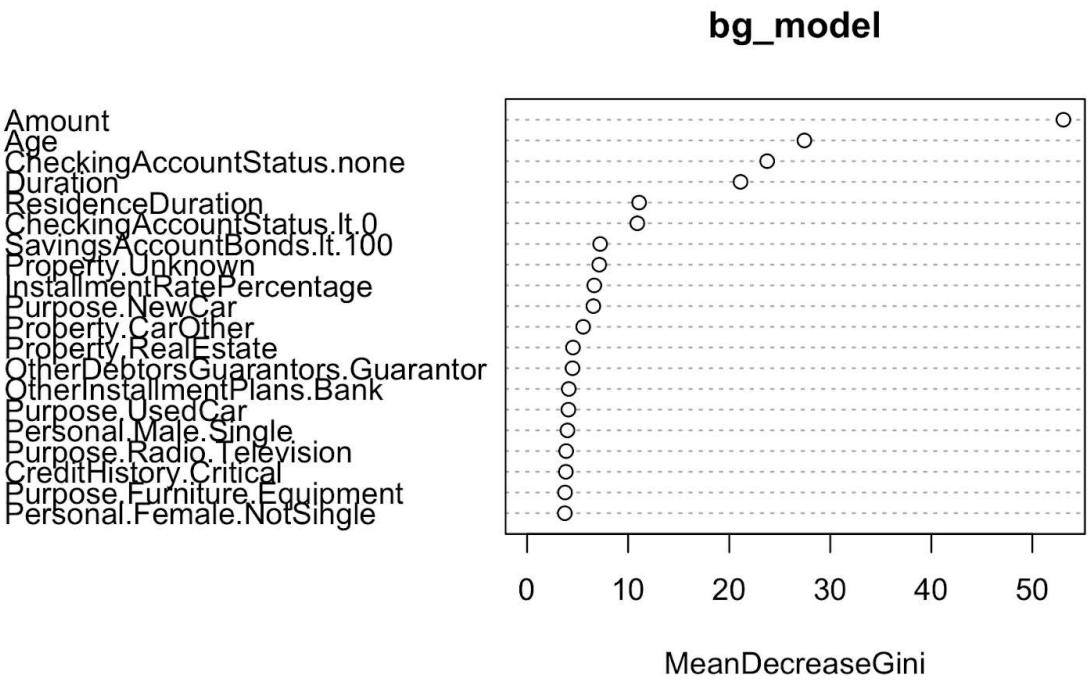
4.6342260	Personal.Female.NotSingle
4.5897480	OtherInstallmentPlans.None
4.5407232	EmploymentDuration.lt.1
4.3700100	Telephone
4.3025670	EmploymentDuration.1.to.4
4.2911994	Job.SkilledEmployee
4.1372485	Personal.Male.Single
4.0902087	Property.RealEstate
3.9896205	Property.CarOther
3.9556674	Purpose.Radio.Television
3.8101040	SavingsAccountBonds.Unknown
3.7401202	Property.Insurance



For Bagging model,

overall	names
53.0704013	Amount

27.4391572	Age
23.7526010	CheckingAccountStatus.none
21.1191978	Duration
11.0788881	ResidenceDuration
10.9081790	CheckingAccountStatus.lt.0
7.2263184	SavingsAccountBonds.lt.100
7.1383858	Property.Unknown
6.6429516	InstallmentRatePercentage
6.5580840	Purpose.NewCar
5.5480119	Property.CarOther
4.5372962	Property.RealEstate
4.4921955	OtherDebtorsGuarantors.Guarantor
4.1186433	OtherInstallmentPlans.Bank
4.0881662	Purpose.UsedCar
3.9925612	Personal.Male.Single
3.8619121	Purpose.Radio.Television
3.8339766	CreditHistory.Critical
3.7536987	Purpose.Furniture.Equipment
3.7330669	Personal.Female.NotSingle
3.5876678	Job.SkilledEmployee
3.5829594	EmploymentDuration.lt.1
3.3406737	CheckingAccountStatus.0.to.200
3.1648817	CreditHistory.NoCredit.AllPaid
3.0873447	NumberPeopleMaintenance
3.0676587	CreditHistory.PaidDuly



d)

Looking at the Feature Importance, one can say that bagging is heavily dependent on Amount with over 50% MeanDecreaseGini, which is over 30% in Random Forest. Considering this, one can say that Random Forest Model is better model for out of sample data, as there might be a possibility that the training data has patterns related to amount which the model might have learnt well, leading to higher variance. This is the reason why random forest is used, where few predictors are not used in all the trees, leading to better overall performance in out of sample data.

The Bagging model, despite having a slightly lower accuracy of 74% compared to Random Forest(77%), demonstrates superior performance in terms of sensitivity, particularly when the positive class is "Bad." This is crucial in banking contexts where accurately identifying "bad customers" is paramount. Random Forest achieves a sensitivity of 0.41 for the "Bad" class, surpassing the sensitivity of 0.43 obtained by Logistic Regression. While Random forest exhibits a slightly lower sensitivity of 0.41 compared to Logistic Regression, Bagging's overall balance between accuracy and sensitivity makes it a more favorable choice for this specific application. In essence, Bagging is able to effectively identify "bad customers" without compromising overall accuracy positions it as the preferred model for banking scenarios where risk mitigation is a primary concern.

Question-3

Part-A

Approach

- **Data Preparation**: Loaded the `credits.csv` dataset into a DataFrame and separated the features from the target variable, classifying the entries as 'Good' or 'Bad'.
- **Data Preprocessing**: To ensure that my numeric features were on a similar scale, standardized them using a method that centered the data around a mean of 0 and a standard deviation of 1. This step is crucial for distance-based algorithms like KNN.
- **Train-Test Split**: Split the dataset into training and testing sets, using 70% of the data for training and 30% for testing to effectively evaluate my model's performance.
- **KNN Classifier Implementation**: Trained KNN classifiers with various values of K = (1, 3, 5, and 10). For each K, I fitted the model to the training data and made predictions on the test set.
- **Model Evaluation**: Generated confusion matrices for each model to visualize the true positives, true negatives, false positives, and false negatives. Additionally, created classification reports that included metrics such as precision, recall, and F1 score for each class. Also calculated and displayed the true positive rate and false negative rate for each KKK.
- **ROC Curve Analysis**: Computed the ROC curves and AUC for each KKK. This involved obtaining the false positive rates and true positive rates. Then these curves to assess model performance across different thresholds.
- **Results Interpretation**: Finally, analyzed the confusion matrices and classification reports to evaluate how well each model performed. Examined the ROC curves to understand the trade-offs between true positive rates and false positive rates.

Code

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, classification_report, roc_curve, auc
from sklearn.preprocessing import StandardScaler

# Load the dataset
df = pd.read_csv('/kaggle/input/credit/Credit.csv')

# Display the dataset before standardization
print("Dataset before standardization:")
```

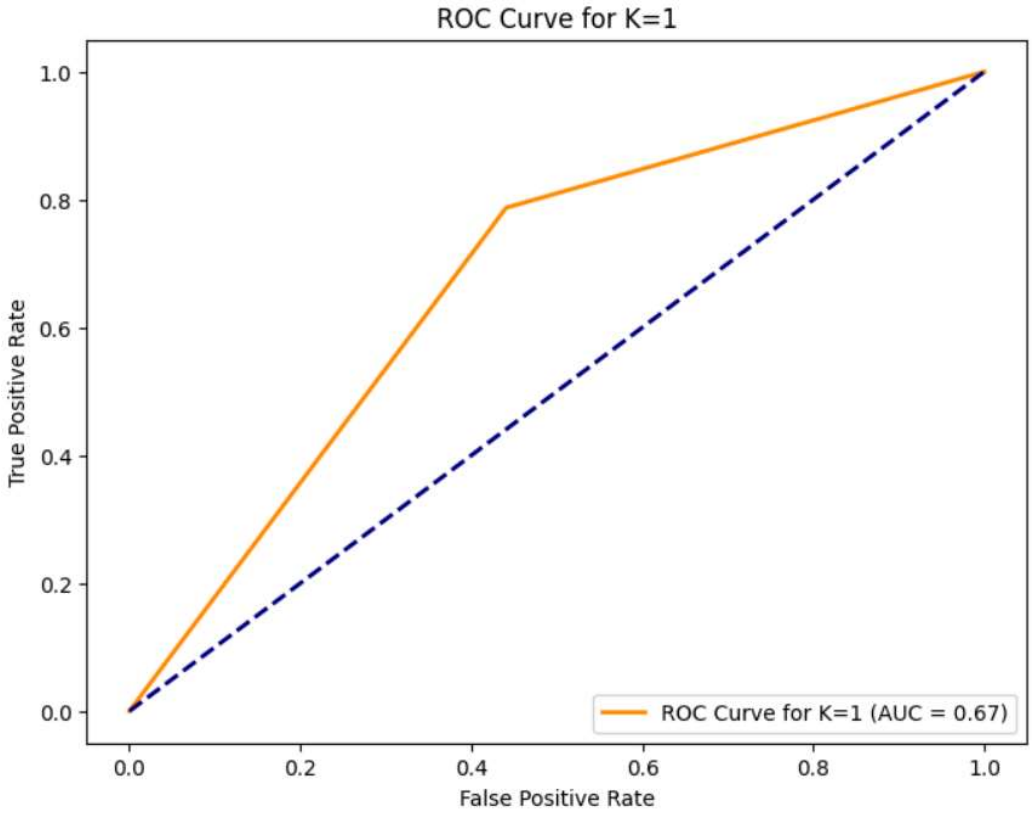
```
print(df.head())

# Select only the numeric columns for standardization (excluding target or
categorical columns if any)
numeric_columns = df.select_dtypes(include=['float64', 'int64']).columns

# Initialize the StandardScaler
scaler = StandardScaler()

# Standardize the numeric columns
df[numeric_columns] = scaler.fit_transform(df[numeric_columns])

# Display the dataset after standardization
print("\nDataset after standardization:")
print(df.head())

target_column = 'Class'

# Prepare features (X) and target (y)
X = df.drop(columns=[target_column])  # Features
y = df[target_column]                 # Target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20,
random_state=42)

k_values = [1, 3, 5, 10]

# Train and test the KNN model for each K value
for k in k_values:
    # Initialize the KNN classifier
    knn = KNeighborsClassifier(n_neighbors=k)

    # Train the model
    knn.fit(X_train, y_train)

    # Predict on the test set
```

```
    y_pred = knn.predict(X_test)

    # Generate confusion matrix
    conf_matrix = confusion_matrix(y_test, y_pred)

    # Calculate True Positive Rate (TPR) and False Negative Rate (FNR)
    TP = conf_matrix[1, 1]  # True Positives
    FN = conf_matrix[1, 0]  # False Negatives
    TPR = TP / (TP + FN) if (TP + FN) > 0 else 0  # True Positive Rate
    FNR = FN / (TP + FN) if (TP + FN) > 0 else 0  # False Negative Rate

    # Print confusion matrix and classification report
    print(f"\nConfusion Matrix for K={k}:")
    print(conf_matrix)

    print(f"\nClassification Report for K={k}:")
    print(classification_report(y_test, y_pred))

    # Include TPR and FNR in the output
    print(f"True Positive Rate for K={k}: {TPR:.4f}")
    print(f"False Negative Rate for K={k}: {FNR:.4f}")

    # ROC Curve analysis
    y_prob = knn.predict_proba(X_test)[:, 1]  # Probability estimates for the
positive class
    fpr, tpr, thresholds = roc_curve(y_test.apply(lambda x: 1 if x == 'Good' else 0),
y_prob)
    roc_auc = auc(fpr, tpr)

    # Plot ROC Curve
    plt.figure(figsize=(8, 6))
    plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC Curve for K={k} (AUC =
{roc_auc:.2f})')
    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'ROC Curve for K={k}')
```

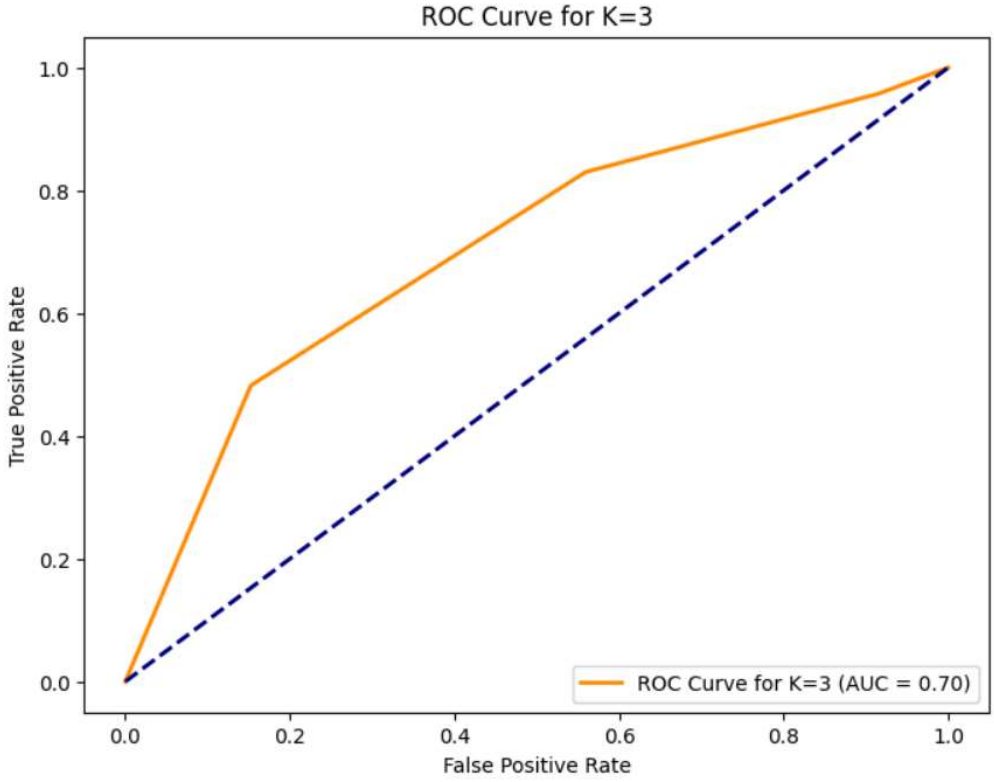
```
plt.legend(loc="lower right")
plt.show()
```

Results

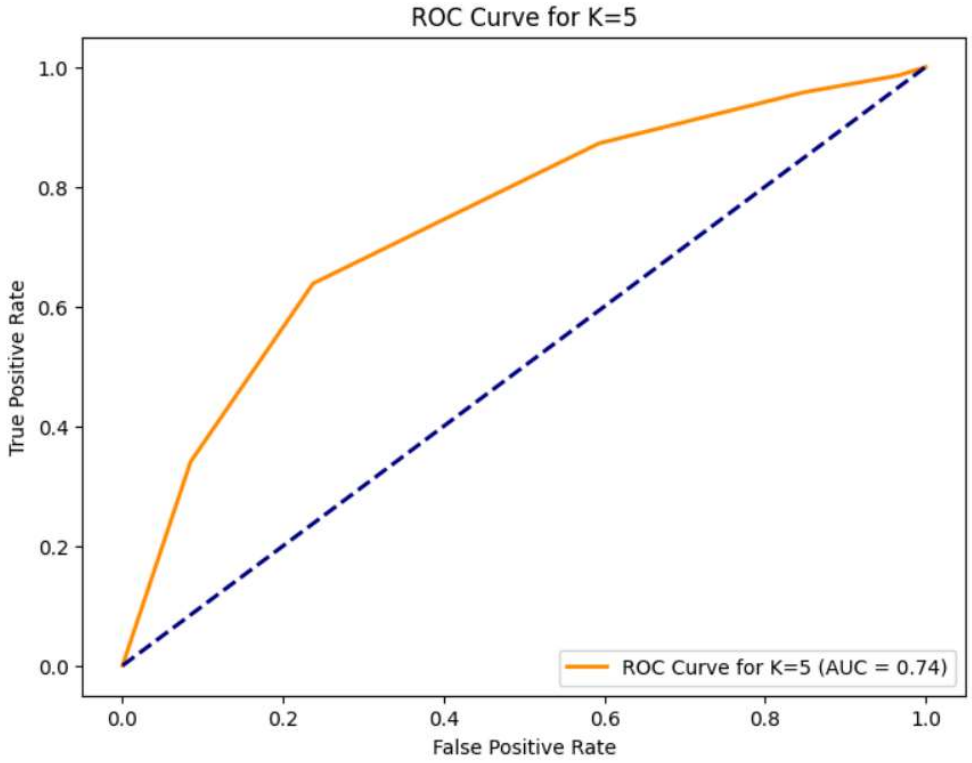
```
Confusion Matrix for K=1:
[[ 33  26]
 [ 30 111]]

Classification Report for K=1:
              precision    recall  f1-score   support

         Bad       0.52      0.56      0.54        59
        Good       0.81      0.79      0.80       141

    accuracy                           0.72       200
   macro avg       0.67      0.67      0.67       200
weighted avg       0.73      0.72      0.72       200

True Positive Rate for K=1: 0.7872
False Negative Rate for K=1: 0.2128
```



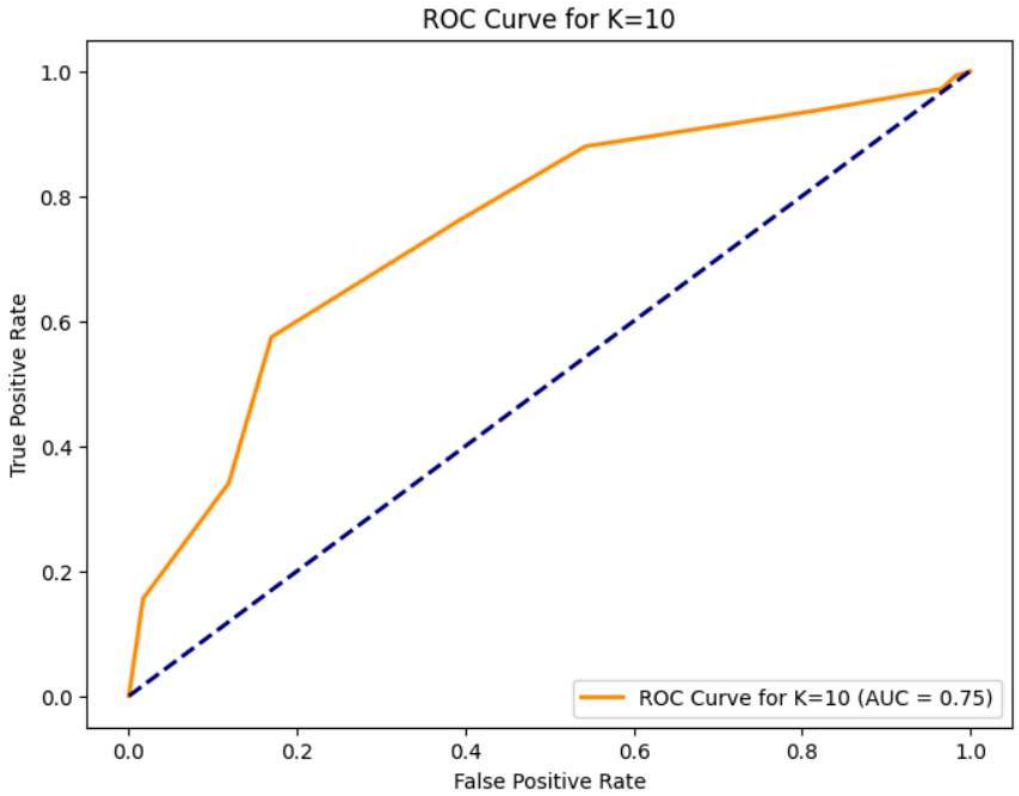
```
Confusion Matrix for K=3:
[[ 26  33]
 [ 24 117]]

Classification Report for K=3:
              precision    recall  f1-score   support

         Bad       0.52      0.44      0.48        59
        Good       0.78      0.83      0.80       141

    accuracy                           0.71       200
   macro avg       0.65      0.64      0.64       200
weighted avg       0.70      0.71      0.71       200

True Positive Rate for K=3: 0.8298
False Negative Rate for K=3: 0.1702
```



```
Confusion Matrix for K=5:
[[ 24  35]
 [ 18 123]]

Classification Report for K=5:
              precision    recall  f1-score   support

         Bad       0.57      0.41      0.48        59
        Good       0.78      0.87      0.82       141

    accuracy                           0.73       200
   macro avg       0.67      0.64      0.65       200
weighted avg       0.72      0.73      0.72       200

True Positive Rate for K=5: 0.8723
False Negative Rate for K=5: 0.1277
```



```
Confusion Matrix for K=10:
[[ 27  32]
 [ 17 124]]

Classification Report for K=10:
              precision    recall  f1-score   support

         Bad       0.61      0.46      0.52        59
        Good       0.79      0.88      0.84       141

    accuracy                           0.76       200
   macro avg       0.70      0.67      0.68       200
weighted avg       0.74      0.76      0.74       200

True Positive Rate for K=10: 0.8794
False Negative Rate for K=10: 0.1206
```



Inference and observation

Throughout the KNN classification process, we observed that the performance of the model remained quite consistent across different values of KKK (1, 3, 5, and 10). Specifically, the confusion matrices, classification reports, and ROC curves revealed that increasing KKK did not result in significant improvements in classification accuracy or other performance metrics. This

indicates that the dataset is likely well-separated, with clear distinctions between the 'good' and 'bad' classes. Since larger values of KKK tend to smooth out the decision boundaries, the fact that performance didn't notably improve suggests that the data is already well-structured, allowing even smaller values of KKK to classify the points accurately. Therefore, increasing K beyond a certain point did not offer any substantial benefit in terms of model performance.

This inference supports the conclusion that the dataset's classes are easily distinguishable, and KNN performs optimally even with smaller values of K.

Part-B

- **KNN Performance**:
  - KNN demonstrated the highest sensitivity among the models, consistently outperforming logistic regression (0.43) and random forest (0.41). This indicates its superior ability to correctly identify positive instances (Good').
  - The model's AUC was also higher than logistic regression's 0.729, meaning KNN was more effective at distinguishing between the 'good' and 'bad' classes across various thresholds, making it the best fit for this dataset.
- **Logistic Regression**:
  - Logistic regression achieved a sensitivity of 0.43, performing slightly better than random forest but still falling short of KNN. This shows that it was reasonably capable of identifying positive cases but not as effective as KNN.
  - With an AUC of 0.729, logistic regression provided a decent overall classification performance, though it was still behind KNN in terms of distinguishing between the two classes.
- **Random Forest (Decision Tree)**:
  - Random forest had a lower sensitivity of 0.41, indicating it struggled to identify positive instances as reliably as KNN and logistic regression. However, its accuracy of 77% reflects its ability to make correct overall classifications.
  - Despite decent accuracy, the model's lower sensitivity implies it was less effective in avoiding false negatives, making it less suitable for cases where identifying the 'Good' class is critical.

Question4

- PartA

Problem Statement:

Fit at least one other binary classifier (e.g., a linear probability model or a Support Vector Machine classifier) to the dataset. Describe its performance relative to the classifiers highlighted above.

Observations:

Able to train a polynomial, which gave excellent results on test data.

```
Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost gamma
  0.1   0.5

- best performance: 0.11

- Detailed performance results:
  cost gamma error dispersion
1  0.1   0.5  0.11 0.04099112
2  1.0   0.5  0.11 0.04099112
3 10.0   0.5  0.11 0.04099112
4  0.1   1.0  0.11 0.04099112
5  1.0   1.0  0.11 0.04099112
6 10.0   1.0  0.11 0.04099112
7  0.1   2.0  0.11 0.04099112
8  1.0   2.0  0.11 0.04099112
9 10.0   2.0  0.11 0.04099112
```

Confusion Matrix Obtained

```
     pred
true   Bad Good
  Bad    72   18
  Good    5  205
```

Confusion Matrix:

		Actual
	Bad	Good
Bad	72	18
Good	5	205

TPR = 0.94
TNR = 0.92

The consistent error rate across different parameter combinations suggests that the SVM model is relatively robust to changes in hyperparameters within the explored range. The model achieves a high overall accuracy of 91%, indicating that it performs well on the given dataset.

The confusion matrix shows a slight class imbalance, with more instances of the "Good" class. This might affect the model's performance, especially in terms of recall for the "Bad" class.
This outperforms previous classifiers which were having around 70% accuracy. This model clearly has better accuracy and True Positive Rate, which is an important metric in financial settings.

Using the Radial kernel, the SVM predicts every observation as Class Good, which might be due to imbalanced data with more number of observations in the good class, and very few observations in the bad class.

- PartB

The data set is unbalanced as we can clearly observe in the data that the number of “Good” Class is much higher than the “Bad” Class. This is also clear from the confusion matrix as it shows pure bias toward the “Good” Class which is present in majority.

The drawbacks of fitting statistical learning technique on unbalanced dataset is :

1. **Bias toward majority class:** The model may learn to predict only the majority class as it was done in our example also when we fit SVM with radial kernel. The best model was biased toward the “Good” (majority) class.
2. **Poor generalization:** The model fails to capture the underlying patterns of the minority class, leading to high false negatives.
3. Performance matrix-like accuracy might be misleading because most of the entries belonging to one class might not give the clear picture.

The confusion matrix can indeed be a good method to evaluate the performance in the cases when the data set is unbalanced as it provides a more detailed view providing false positives, false negatives, and true positives. However, it should be complemented with other metrics like precision, recall, and F1 score, especially in unbalanced scenarios.

To address the concern of unbalanced data set we can try to take the following steps:
We can use a resampling approach to try and balance the data set. This can be done by the following 2 approaches

- Oversampling the Minority class
- Removing some of the samples from the majority class can also balance the data but it also decreases the useful information.

Some other methods to address unbalance data is using techniques like :

1. Cost-Sensitive learning: Modifying the algorithm to account for misclassification costs can make the classifier more sensitive to the minority class.
2. Anomaly Detection Techniques: Treating the minority class as anomalies can also be effective in certain contexts. Techniques like isolation forest
3. Also using Ensemble methods (like random forest) which will use multiple models for anomaly detection can also help improving performance

I think these methods could work because these methods aim to ensure that the model pays equal attention to both classes, helping to improve recall and precision for the minority class. By providing a more balanced view of the data, the classifier can better learn the patterns associated with each class.