MBA786M: Alert Models in Finance

Project Report

Introduction

Dataset Overview

- **Dependent Variable**: “Class” (Good or Bad)
- **Independent Variables**:
    - Checking account status
    - Loan duration
    - Credit history
    - Loan amount
    - Employment duration
    - Age, etc.

Imported Libraries

Various types of libraries we have imported to perform important and essential tasks to build and evaluate ML models given in the project.

- **Pandas** and **NumPy** are used to manipulate the data and apply numerical operations to the given data.
- **Scikit-learn** provide tools for our applied model and algorithms like **Logistic Regression**, **Decision Trees**, **Bagging**, **Random Forest**, and **K-Nearest Neighbors** (KNN), along with functions and tools for model evaluation, such as accuracy scores, confusion matrices, ROC curves, and AUC metrics.
- **Matplotlib** and **Seaborn** help to visualise data , whereas **LabelEncoder** and **StandardScaler** handles how to preprocess data and how to standardise it.

(Task 1(a)) Logistic Regression: Full Dataset

Model Fitting

We started initially by fitting a ML model of a **logistic regression** to our entire given dataset in the provided excel sheet. The variable “Class” which had value “Good “ or “Bad” was taken as a dependent variable and the variable column was encoded as 1 for "Bad" credit and 0 for "Good" credit. We have used a pipeline called StandardScaler() to standardise the input features of all the given 61 columns and the LogisticRegression() tool helped us to perform the classification.

The logistic model was trained on the entire dataset of the excel using the following parameters:

- **Solver**: liblinear which was chosen for its compatibility with small datasets and binary classification tasks
- **Max Iterations**: 200 was chosen to ensure convergence.

Varying Probability Thresholds

In the logistic regression model , 0.5 is taken as the default threshold for classifying probabilities, which means that any customer with a probability greater than or equal to 0.5 is classified as having "Bad". However, varying the threshold can significantly change our model's performance and accuracy. We have evaluated our model using three different thresholds: **0.2**, **0.35**, and **0.5**, by computing a confusion matrix for each case.

Confusion Matrices and Performance Metrics

1. **Threshold = 0.2**

   Confusion Matrix obtained:

   ```
   [[44 5]

   [256 695]]
   ```

   - True Positive Rate (TPR): 0.90
   - False Positive Rate (FPR): 0.27
   - Accuracy: 0.739

At this threshold of 0.2, our model is quite lenient, which classified a larger number of given customers as "Bad" . While the TPR at this threshold is high which indicates that our model captured most of the actual "Bad" customers. But the main thing is that the FPR is also quite high which also indicates a good number of misclassifications of "Bad" customers as "Good".

2. **Threshold = 0.35**

   Confusion Matrix obtained:

   ```
   [[100 27]

    [200 673]]
   ```

   - True Positive Rate (TPR): 0.79
   - False Positive Rate (FPR): 0.23
   - Accuracy: 0.773

As the threshold was increased to 0.35, the TPR decreased to 0.79 which means that fewer "Bad" customers are identified correctly as compared to before, but on the other hand FPR also decreased which means that only fewer "Bad" customers are incorrectly classified . This threshold of 0.35 offered us a good balance between the TPR and FPR . And overall as it is evidenced that there is a slight increase in overall accuracy to 0.773.

3. **Threshold = 0.5**

   Confusion Matrix obtained:
   ```
   [[160 75]

    [140 625]]
   ```

   - TPR: 0.68
   - FPR: 0.18
   - Accuracy: 0.785

At the threshold of 0.5, our model becomes more conservative in classifying customers as having "Bad". The TPR dropped further to 0.68, but the FPR improved to 0.18. The overall accuracy increases to 0.785. This threshold gives us a more traditional balance between TPR and FPR, by classifying fewer customers as "Bad" but with a lowered rate of false positives and overall increased accuracy.

Model Selection

Each of the thresholds offers us a different trade-off between TPR and FPR. Lower thresholds capture more TPR but on the other side give high FPR , and as we use higher thresholds it reduces FPR but at the cost of decreasing our model's TPR. Based on the results we obtained, a threshold of **0.35** seems to offer us the great balance between high TPR and low FPR. However, the final choice of what threshold to use would solely depend upon the objective of our business whether it is more important for them to minimise false positives or to maximise true positives. If it requires a balance between TPR and FPR then we will choose a model with 0.35 as threshold.

(Task 1(b)) Logistic Regression: Training and Test Split

To evaluate how well and good our logistic regression model performs on unseen data, we have splitted the given dataset into a **training set (70%)** and a **test set (30%)**. The model was first trained by us on the training set and then we tried to evaluate it on the test set by using the same three thresholds.

Model Performance on Test Set

We have just repeated the confusion matrix and performance analysis for the test set.

1. **Threshold = 0.2**

   Confusion Matrix obtained:
   ```
   [[ 10  2]

    [ 74 214]]
   ```

- TPR: 0.83
- FPR: 0.26
- Accuracy: 0.747

The model seemed to be lenient, by achieving a high TPR and correctly classifying 83% of "Bad" credit customers, but we got a relatively high FPR.

2. **Threshold = 0.35**

   Confusion Matrix obtained:
   ```
   [[ 22  10]

    [ 62 206]]
   ```

   - TPR: 0.69
   - FPR: 0.23
   - Accuracy: 0.760

This threshold results in a balanced model performance. The TPR drops, but the FPR also reduces slightly, making it good for decision-making.

3. **Threshold = 0.5**

   Confusion Matrix obtained:
   ```
   [[ 33  27]

    [ 51 189]]
   ```

   - TPR: 0.55
   - FPR: 0.21
   - Accuracy: 0.740

At the default threshold of 0.5, the model behaves more conservative, with a lower TPR but also a lower FPR, making it a reasonable threshold depending on the application.

Model Selection

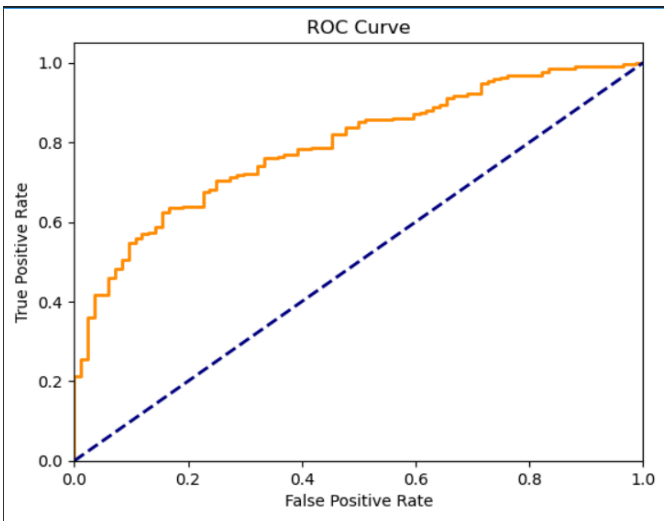
On the test set, the threshold of **0.35** also performs well, it offers a good balance between the TPR and FPR with an accuracy of **0.760**. As we have seen, the threshold of 0.5 offers slightly better accuracy on the full dataset without splitting , but it seemed to be less effective on the test set as its accuracy dropped to 0.740. Hence we will choose a model with 0.35 as a threshold value.

(Task 1(c)) ROC Curve and AUC

To gain more deep insights of our model's performance , we have plotted the **Receiver Operating Characteristic (ROC) curve** and calculated the **Area Under the Curve (AUC)** for the test set.

ROC Curve Interpretation

The ROC curve gives us and shows us the trade-off between the TPR and FPR at various classification thresholds. The closer the curve is to the top-left corner, the better the model is at distinguishing between the two classes.



AUC Metric

The **AUC** for the logistic regression model on the test set is **0.79**. AUC values range between 0.5 and 1.0 where 1 means perfect classification. An AUC of 0.79 indicates that the model has good classification power.

(Task 2(a)) Fitting Classification Tree, Bagging, and Random Forest Models

1. Decision Tree Model

A Decision Tree was fitted using a maximum depth of 100. The tree becomes very deep, and the model tries to perfectly predict all the data points.

Confusion Matrix obtained:

```
[[300   0]

 [0 700]]
```

- Accuracy: 100%

2. Bagging Classifier

The Bagging Classifier is an ensemble method in which we have trained 100 Decision Trees on random subsets of the data . We took a random 66% of the dataset in each iteration as a sample . As we know that Bagging reduces overfitting by averaging.

Confusion Matrix obtained:

```
[[297   0]

 [ 3 700]]
```

- Accuracy: 99.7%

3. Random Forest Model

Random Forest is another ensemble method with additional randomness in selecting features for each tree.

Confusion Matrix obtained**:**

```
[[296  0]

 [ 4 700]]
```

- Accuracy: 99.6%

4. Overfitting Indicators

1) The Decision Tree model achieved perfect accuracy of 100%, which strongly indicates overfitting, as its high depth likely captures every detail of the training data, reducing its ability to generalise to unseen data.

2) The Bagging model showed slight misclassification with three incorrect predictions but still displayed very high accuracy, suggesting some level of overfitting due to training and testing on the same dataset.
3) Similarly, the Random Forest model had only four misclassifications, giving us near-perfect accuracy, and while it generalises slightly better than the Decision Tree, its performance is likely overestimated because it was also evaluated on the training data.

(Task 2(b)) Model Evaluation on Split Dataset

In this task, the dataset was split into training (70%) and test (30%) sets. The performance of three models—Decision Tree, Bagging Classifier, and Random Forest—was evaluated on the test set.

1. Decision Tree

Confusion Matrix obtained:

```
[[ 41  41]

 [ 43 175]]
```

- **True Positive Rate (TPR):** 0.50
- **False Positive Rate (FPR):** 0.20
- **Accuracy:** 0.720

The Decision Tree model gives an average performance with a TPR of 0.50 and an accuracy of 72% along with exhibiting a high misclassification rate.

2. Bagging Classifier

Confusion Matrix obtained:

```
[[ 42 16]

  [42 200]]
```

- **True Positive Rate (TPR):** 0.72
- **False Positive Rate (FPR):** 0.17
- **Accuracy:** 0.807

The Bagging Classifier performs better than the Decision Tree, with a TPR of 0.72 and an accuracy of 80.7%. It also reduces the misclassification, leading to better overall generalisation.

3. Random Forest

Confusion Matrix obtained:

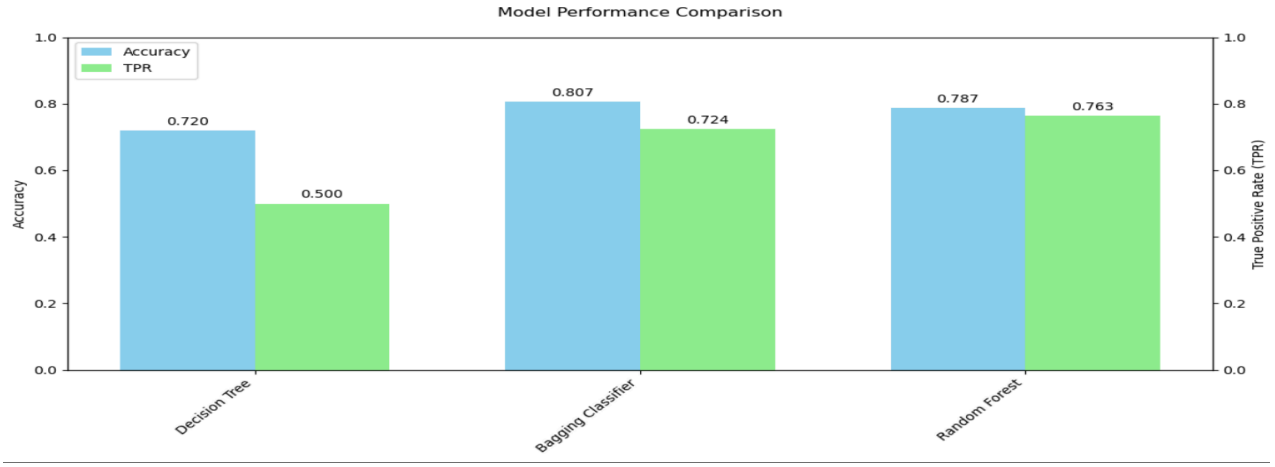
```
[[ 29  9]

 [55 207]]
```

- **True Positive Rate (TPR):** 0.76
- **False Positive Rate (FPR):** 0.21
- **Accuracy:** 0.787

The Random Forest model performs well with a highest TPR of 0.76 and giving an accuracy of 78.7% which is slightly less than Bagging. It reduces false negatives but shows higher false positive rate as compared to the Bagging model, indicating some trade-offs.

Visualisation

The performance comparison between different models is shown in a bar plot, showing the accuracy & TPR for each model.



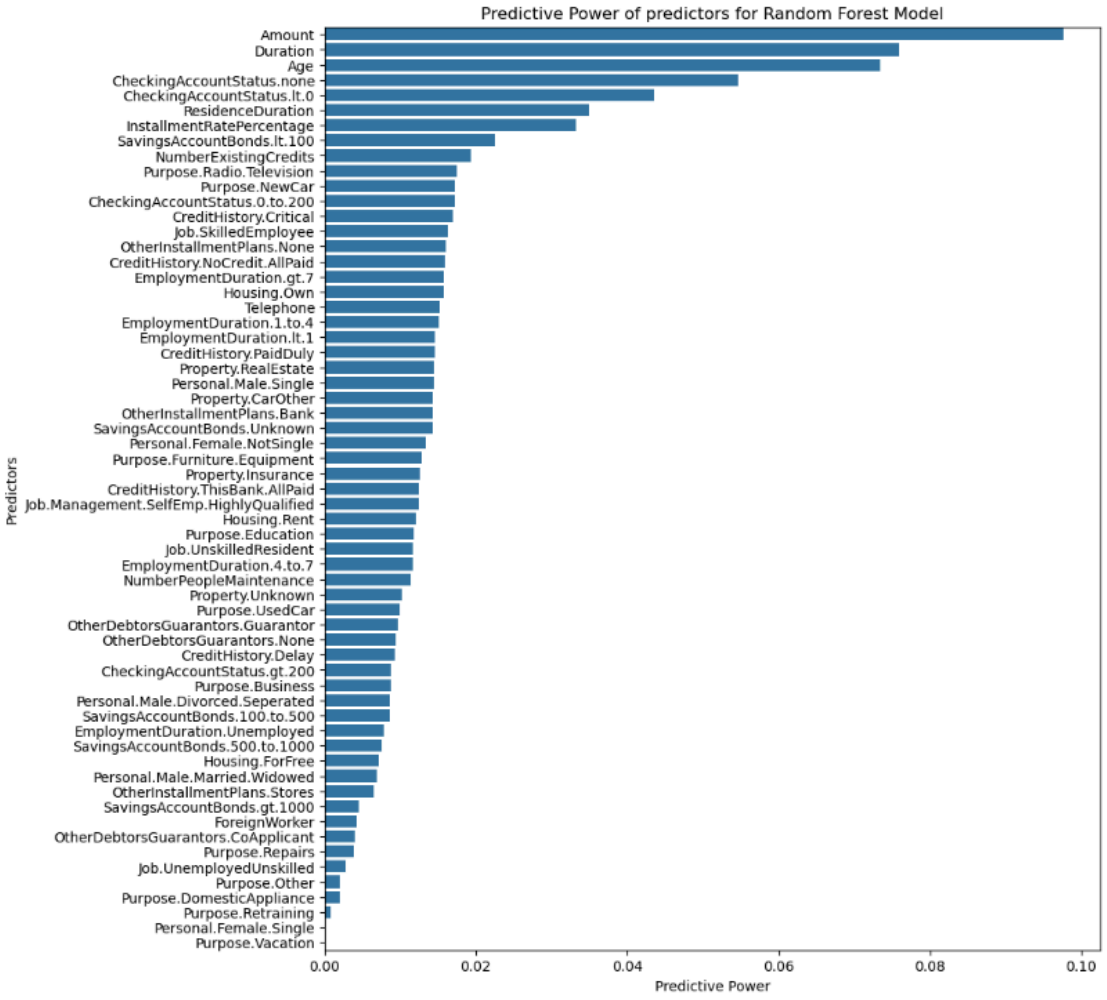
Model Selection

Among the three models, the **Bagging Classifier** performs well on test data as it gives highest accuracy (80.7%) and better balance between true positive and false positive rates. The **Random Forest** is also a good option with a bit higher TPR but lower accuracy. However, the Bagging model gives a good trade-off between accuracy and misclassification error rates, making it the better model for this task.

(Task 2(c)) Ranking Predictors by Predictive Power

The **Random Forest** model that is chosen for its highest accuracy (78.7%) and TPR (0.763), showed the following key predictors:

Amount (predictive power : 0.0976) was the most important, followed by **Duration** (predictive power : 0.0759) and **Age** (predictive power : 0.0734). A bar plot is plotted which visually demonstrates these rankings, highlighting us the impact of significant features.



(Task 2(d)) Comparison of Model Performance

- The **Bagging Classifier** gave us the best performance among all the models that we have analysed yet, with the highest accuracy (80.7%) and TPR (0.72).
- The **Random Forest** model also provided a good competitive performance with a good TPR of 0.76 and comparable accuracy of 78.7%, making it a strong candidate for classification tasks.
- The models from Question 1 which had a threshold of 0.35, exhibited a good reasonable performance with a good balance between TPR (0.69) and FPR and giving a accuracy of 76% but generally lag behind from the Bagging Classifier and Random Forest in terms of accuracy and overall predictive power.

(Task 3(a)) K-Nearest Neighbors (KNN) Classifier Performance

In this task we are going to evaluate the performance of the K-Nearest Neighbors (KNN) classifier after standardising the given predictors. The classification is generally performed with different values of *K* (1, 3, 5, 10) on the test set. Then we try to observe that how varying *K* impacts our model's performance. Then we assess the performance of it using metrics such as the True Positive Rate (TPR), False Positive Rate (FPR), accuracy, and confusion matrices.

Data Splitting and Standardization

We have initially done the splitting of given dataset to training and test sets using a 70-30 ratio (train_test_split with test_size=0.3 and random_state=97).

The predictors are then standardised using the library function called StandardScaler. This function tries to ensure that all features have a mean = 0 and standard deviation = 1. This type of standard scaling is important and necessary for KNN as it is sensitive to feature magnitude.

KNN Classifier Fitting

KNN classifiers are fitted to the standardised training data with varying values of *K* (1, 3, 5, and 10).

Then we evaluated our model's prediction on the test set using a confusion matrix, and performance metrics (accuracy, TPR, and FPR).

Model Performance on Test Set

1. **K=1:**

Confusion Matrix obtained:

```
[[ 39  48]

 [ 45 168]]
```

- True Positive Rate (TPR): 0.45
- False Positive Rate (FPR): 0.21
- Accuracy: 0.6900

With *K=1*, the model TPR indicates that 45% of the positive cases are correctly classified. The accuracy is relatively low at 69%. The model might be overfitted because of this low *K* value.

2. **K=3:**

Confusion Matrix obtained:

```
[[ 33  38]

 [ 51 178]]
```

- True Positive Rate (TPR): 0.46
- False Positive Rate (FPR): 0.22
- Accuracy: 0.7033

At *K=3*, both accuracy (70.33%) and TPR (46%) gets slightly improved. The model begins to generalise better by reducing the impact of individual noisy data points. However,we can see that the FPR also increases marginally slightly to 0.22.

3. **K=5:**

Confusion Matrix obtained:

```
[[ 29  37]

 [ 55 179]]
```

- True Positive Rate (TPR): 0.44
- False Positive Rate (FPR): 0.24
- Accuracy: 0.6933

At *K=5*, the accuracy decreases to 69.33%, and the TPR has dropped to 44%. The FPR rises to 0.24. This result indicates that the model performs similarly to *K=1* as it is also struggling slightly to classify the positive cases correctly at this value of *K*.

4. **K=10:**

Confusion Matrix obtained:

```
[[ 38  35]

 [ 46 181]]
```

- True Positive Rate (TPR): 0.61
- False Positive Rate (FPR): 0.20
- Accuracy: 0.7633

With *K=10*, the model achieves its best performance, with an accuracy of 76.33% and a TPR of 61%. The FPR remains relatively low at 0.20.

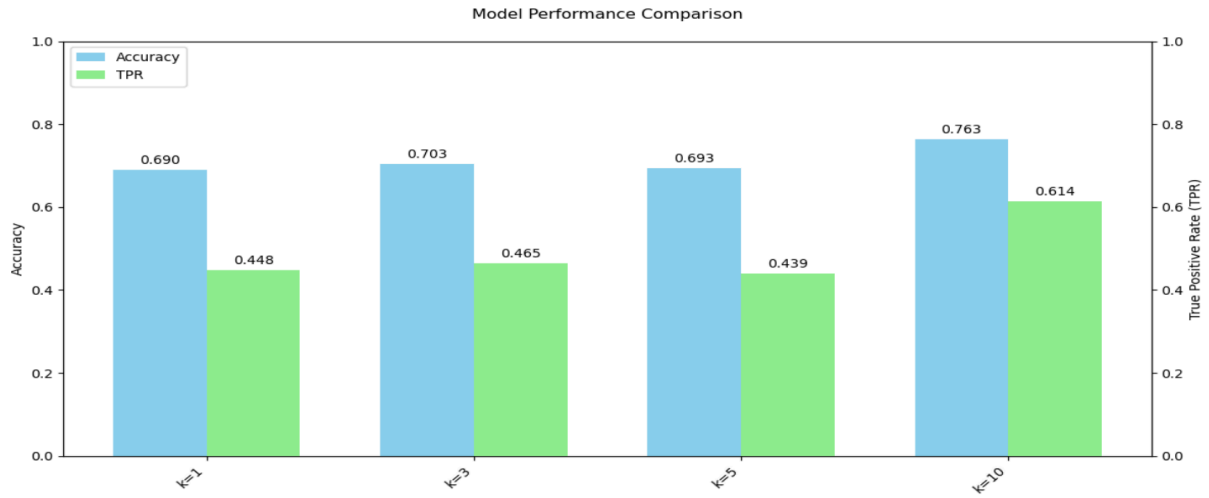
Analysis

The results indicated to us that as we increase the value of K , the accuracy and TPR tend to improve. For *K=1*, the model had low TPR (0.45) and an accuracy of 0.6900. But when we increased to *K=10*, both the accuracy (0.7633) and TPR (0.61) showed a great noticeable improvement in results. This suggests and indicates that higher *K* values allow the model to reduce overfitting.

However, we also saw that on increasing *K* there is a slight increase in the FPR, though the changes are very minimal which can be almost neglected in front of such remarkable increase in TPR and accuracy . Overall, the classifier performs best with *K=10*, achieving the highest accuracy and TPR while maintaining a relatively low FPR.

Visualisation

The performance comparison between the different models is illustrated in a bar plot, showing the accuracy and TPR for each *K* value.



Comparison of Model Performance (Task 3(b))

- The logistic regression model stands as a good alternative with having an accuracy = 76% and TPR = 0.69 but it does not surpass the performance given by ensemble methods.
- The **Bagging Classifier** in Question 2 has achieved the highest accuracy = 80.7% and a TPR = 0.72, outperforming all KNN models.
- **Random Forest** also performed very well with 78.7% accuracy and TPR 0.76 and demonstrated a robust predictive capabilities of it.
- Among all of the KNN models, **K=10** got the highest accuracy (76.33%) and TPR (0.61), but this model still lags behind the tree-based models in terms of accuracy and low TPR as compared to tree based models.
- Overall, the tree-based models i.e Bagging and Random Forest have shown an excellent performance compared to the KNN models, and the logistic regression model, particularly in balancing TPR and FPR

(Task 4(a)) Fitting other Models

In this task we are going to evaluate SVMs on two kernels - Linear and Radial. For both of these we have used grid-search for tuning the hyper parameters. Then we try to evaluate the confusion matrix and accuracy of both of these models.

Parameters used for Linear SVM -

```
'C': [0.001, 0.01, 0.1, 1, 5, 10, 100]
```

Parameters used for Radial SVM -

```
'C': [0.1, 1, 10, 100, 1000]
```

```
'gamma': [0.5, 1, 2, 3, 4]
```

Data Splitting and Standardization

We have initially done the splitting of given dataset to training and test sets using a 70-30 ratio (train_test_split with test_size=0.3 and random_state=97).

The predictors are then standardised using the library function called StandardScaler. This function tries to ensure that all features have a mean = 0 and standard deviation = 1. This type of standard scaling is important and necessary for KNN as it is sensitive to feature magnitude.

Model Performance on Test set

1. **Linear SVM**

   Confusion Matrix obtained:

   ```
   [[ 49  27]

    [ 40 184]]
   ```

   - True Positive Rate (TPR): 0.64
   - False Positive Rate (FPR): 0.18
   - Accuracy: 0.777

Best Hyperparameters found from gridsearch are `'C' = 0.001`

Best cross - validation score : 0.7375

The model has good accuracy but its TPR is very low. It will approximately miss out on 36% of the bad credit risks. Hence Logistic Regression is a better model in comparison to this. It performs poor than the Bagging and Random forest method but is comparable to the KNN model with K=10 and Logistic regression model.

2. **Radial SVM**

   Confusion Matrix obtained:

```
[[ 0  0]
 [ 89 211]]
```

   - True Positive Rate (TPR): 0.0
   - False Positive Rate (FPR): 0.30
   - Accuracy: 0.703

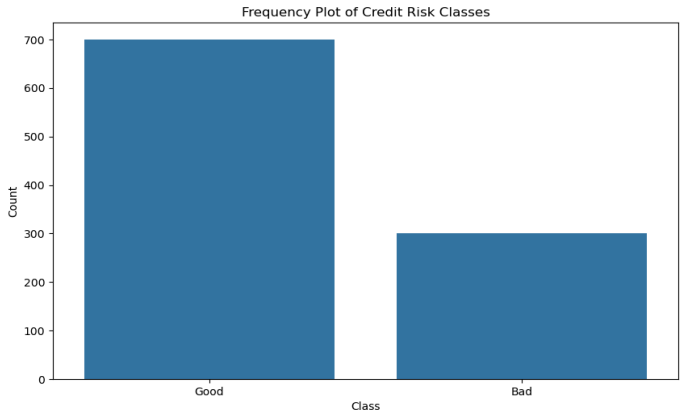
Best Hyperparameters found from gridsearch are `'C' = 0.1 'gamma' = 0.5`

Best cross - validation score: 0.6986

The model has a good accuracy but its TPR is null. Showing that it misses out on all 'Bad' Credit risk classifications. Hence it would perform very poorly in deployment.
One of the reason for such bad performance may be that there is no radial decision boundary that can classify the data at hand.This perform very poor as compared to other models deployed above.

(Task 4(b)) Handling Data Imbalance

Here we explore the distribution of the target variable amongst the data points. We can see that 30% of the data points belong to the 'Bad' Credit risk class. This means that even if the model is highly biased and ends up predicting 'Bad' class for all the test data points, we would get a 30% accuracy. Hence our training dataset is not balanced.



Fitting a statistical model on an unbalanced dataset often leads to bias toward the majority class, resulting in misleading accuracy and poor performance in predicting the minority class. Models may achieve high accuracy by predicting mostly majority class labels, but they fail to capture the minority class, leading to poor recall. This distorts decision boundaries and makes metrics like accuracy unreliable in assessing true model performance.
We can use a confusion matrix to see how the model performs on both 'Positive' class data and 'Negative' class data. A high TPR and low FPR is desirable. Yes, a confusion matrix is a

useful performance metric for imbalanced datasets as it provides detailed insights into how well the model classifies both classes, including the minority class.

Some of the techniques that can be used to handle unbalanced dataset are :-

1) **Undersampling :** It is a technique that reduces the size of the majority class by removing examples to balance the dataset. It works because it prevents the majority class from dominating the model.
2) **Oversampling :** It is a technique that increases the size of the minority class by duplicating or generating more examples. It works because it gives the minority class more representation during training.
3) **SMOTE analysis :** It is a technique that generates new synthetic examples for the minority class by interpolating between existing ones. It works because it adds diversity to the minority class without duplication, improving model learning.

We have used SMOTE analysis for scaling the X_train and y_train before fitting Logistic Regression and LinearSVM models. It improves model performance on imbalanced data by providing more representative training data for both classes, which can result in better classification and reduced overfitting or underfitting.